

Анализ эффективности векторизирующих компиляторов на архитектурах Intel 64 и Intel Xeon Phi

Ольга Владимировна Молдованова ^{1,2}

ovm@sibguti.ru, ovm@isp.nsc.ru

Михаил Георгиевич Курносов ^{1,2}

WWW: www.mkurnosov.net

¹ Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики, Новосибирск

² Лаборатория вычислительных систем
Институт физики полупроводников им. А.В. Ржанова СО РАН, Новосибирск

*Всероссийская научная конференция памяти А.Л. Фуксмана «Языки программирования и компиляторы» (PLC-2017)
г. Ростов-на-Дону, 3-5 апреля 2017 г.*

Векторные вычислительные системы и процессоры



1970 – 1990 гг.

- Векторно-конвейерные процессоры и системы

- CDC STAR-100, CYBER-203, CYBER-205
- Cray 1, Cray X-MP, Cray Y-MP, NEC SX, IBM ViVA, Fujitsu FACOM VP, Hitachi HITAC S-810

- Длинные векторные регистры
- Ускорение x64-128 раз

1995 – н. в.

- SIMD-процессоры (наборы векторных инструкций)

- Intel MMX/SSE/AVX
- IBM AltiVec
- ARM NEON SIMD
- MIPS MSA

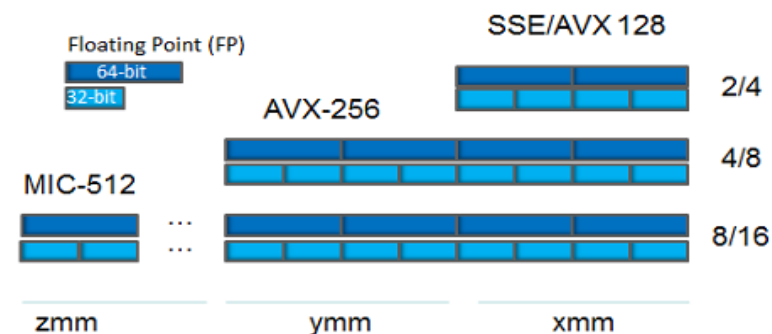
- Короткие векторные регистры
- Ускорение x2-64 раза

SIMD-инструкции процессоров

▪ Наборы векторных инструкций

- Intel MMX/SSE/AVX/AVX-3, AVX-512
- IBM AltiVec
- ARM NEON SIMD
- MIPS MSA

Векторные регистры (Intel 64, Intel Xeon Phi)



▪ Достижимые ускорения

Тип данных	Intel SSE (регистры 128 бит)	Intel AVX (регистры 256 бит)	Intel AVX-512 (регистры 512 бит)	ARMv8 Scalable Vector Extension (регистры 128-2048 бит, RIKEN Post-K supercomputer, 2020)
double	x2	x4	x8	x32
float	x4	x8	x16	x64
int	x4	x8	x16	x64
short int	x8	x16	x32	x128

▪ Причины снижения ускорения

- Адреса массивов не выравнены на заданную границу (32 байта для AVX и 64 байта для AVX-512)
- Смешанное использование SSE- и AVX-инструкций (AVX-SSE Transition Penalties) [1]

[1] Konsor P. *Avoiding AVX-SSE Transition Penalties* // URL: <https://software.intel.com/en-us/articles/avoiding-avx-sse-transition-penalties>

Способы векторизации кода

Полный контроль,
низкая переносимость

Простота использования,
высокая переносимость

- **Ассемблерные вставки**
- **Интринсики (intrinsics)** –
встроенные функции и типы данных компилятора
- **SIMD-директивы** компиляторов,
стандартов OpenMP, OpenACC
- **Языковые расширения** (Intel Array Notation, Intel ISPC, Apple Swift SIMD)
и библиотеки (C++17 SIMD, Boost.SIMD, SIMD.js)
- **Автоматическая векторизация компилятором**

```
void add_sse(float *a, float *b, float *c) {  
    __asm__ __volatile__ (  
        "movaps ([a]), %%xmm0 \n\t"  
        "movaps ([b]), %%xmm1 \n\t"  
        "addps %%xmm1, %%xmm0 \n\t"  
        "movaps %%xmm0, [c] \n\t"  
    )  
}
```

```
void add_sse(float *a, float *b, float *c) {  
    __m128 t0, t1;  
    t0 = _mm_load_ps(a);  
    t1 = _mm_load_ps(b);  
    t0 = _mm_add_ps(t0, t1);  
    _mm_store_ps(c, t0);  
}
```

```
void f(double *a, double *b, double *c, int n) {  
    #pragma omp simd  
    for (int i = 0; i < n; i++)  
        c[i] += a[i] * b[i];  
}
```

```
$ gcc -ftree-vectorize ./vec.c  
vec.c:13:5: note: loop vectorized  
vec.c:18:5: note: not vectorized, possible dependence between data-refs
```

Цель работы

- Определение основных видов циклов, автоматическая векторизация которых компиляторами Intel C/C++, PGI C/C++, GNU GCC, LLVM/Clang на архитектурах Intel 64 и Intel Xeon Phi затруднена
- Оценка времени выполнения и ускорения векторизованных циклов

Набор тестовых циклов

Векторные ВС:
Cray, NEC, IBM,
DEC, Fujitsu, Hitachi

**TSVC – Test Suite for
Vectorizing Compilers [1]**
(122 цикла на Fortran)

*Наборы векторных
инструкций:*
Intel SSE/AVX, IBM AltiVec,
ARM NEON SIMD, MIPS MSA

**ETSVC – Extended Test Suite
for Vectorizing Compilers [2, 3]**
(151 цикл на C)

1991

2011

Категории циклов ETSVC

Категория	Число циклов
Анализ зависимостей по данным (dependence analysis)	36
Анализ потока управления и трансформация циклов (vectorization)	52
Распознавание идиоматических конструкций (idiom recognition)	27
Полнота понимания языка программирования (language completeness)	23
Контрольные циклы (control loops)	13

[1] Levine D., Callahan D., Dongarra J. *A Comparative Study of Automatic Vectorizing Compilers* // Journal of Parallel Computing. 1991. Vol. 17. pp. 1223–1244.

[2] Maleki S., Gao Ya. Garzarán M.J., Wong T., Padua D.A. *An Evaluation of Vectorizing Compilers* // Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT-11), 2011. pp. 372–382.

[3] *Extended Test Suite for Vectorizing Compilers*. URL: <http://polaris.cs.uiuc.edu/~maleki1/TSVC.tar.gz>

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000

#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif

__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];

int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

Каждый цикл – отдельная функция (всего 151 функция)

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000

#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif

__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];

int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000

#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif

__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];

int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Выравнивание адресов массивов на заданную границу

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Выравнивание адресов массивов на заданную границу

Инициализация массивов значениями, характерными для теста

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Выравнивание адресов массивов на заданную границу

Инициализация массивов значениями, характерными для теста

Увеличение времени выполнения теста (формирование статистики)

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Выравнивание адресов массивов на заданную границу

Инициализация массивов значениями, характерными для теста

Увеличение времени выполнения теста (формирование статистики)

Предотвращение нежелательной оптимизации внешнего цикла

Пример цикла из тестового набора ETSVC

```
#define TYPE float
#define LEN (125 * 1024 / sizeof(TYPE))
#define LEN2 256
#define ntimes 200000
```

```
#ifdef MIC
    #define ALIGN 64
#else
    #define ALIGN 32
#endif
```

```
__attribute__((aligned(ALIGN))) TYPE X[LEN], Y[LEN], Z[LEN], U[LEN], V[LEN];
__attribute__((aligned(ALIGN))) TYPE aa[LEN2][LEN2], bb[LEN2][LEN2], cc[LEN2][LEN2];
```

```
int s000() {
    init("s000 ");
    clock_t start_t = clock();
    for (int nl = 0; nl < 2 * ntimes; nl++) {
        for (int i = 0; i < LEN; i++)
            X[i] = Y[i] + 1;
        dummy((TYPE*)X, (TYPE*)Y, (TYPE*)Z, (TYPE*)U, (TYPE*)V, aa, bb, cc, 0.0);
    }
    clock_t end_t = clock();
    printf("S000\t %.2f \t\t", (double)((end_t - start_t)/1000000.0));
    check(1);
    return 0;
}
```

TYPE – тип данных массивов: double, float, int, short int

LEN, LEN2 – размеры одномерных и двумерных массивов

ntimes – количество повторений внешнего цикла

Выравнивание адресов массивов на заданную границу

Инициализация массивов значениями, характерными для теста

Увеличение времени выполнения теста (формирование статистики)

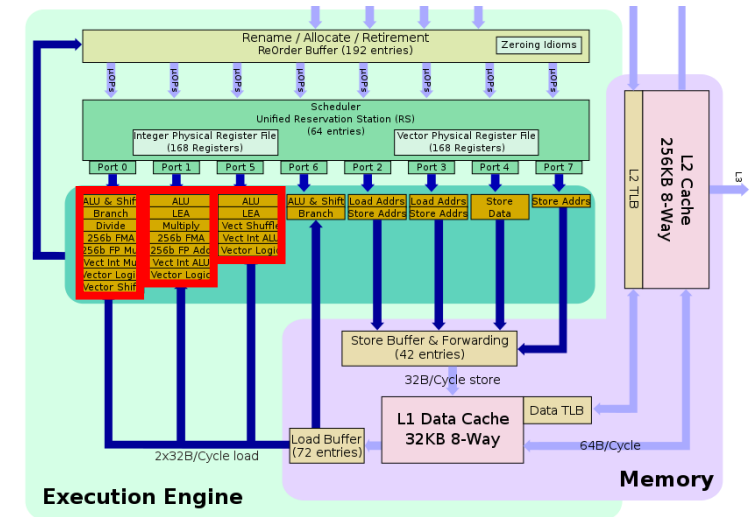
Предотвращение нежелательной оптимизации внешнего цикла

Вычисление контрольной суммы элементов итогового массива

Целевые архитектуры: Intel 64 и Intel Xeon Phi

■ Двухпроцессорный NUMA-сервер

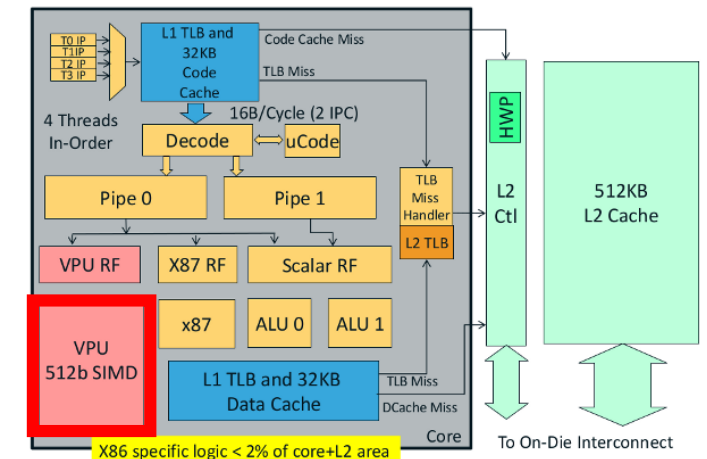
- **2 процессора Intel Xeon E5-2620 v4:**
архитектура Intel 64, микроархитектура Broadwell, 8 ядер,
Hyper-Threading включен, группа векторных АЛУ с поддержкой AVX 2.0
- **Память:** 64 GiB, DDR4,
- **Операционная система:** GNU/Linux CentOS 7.3 x86-64
(ядро linux 3.10.0-514.2.2.el7)



<https://en.wikichip.org/wiki/intel/microarchitectures/broadwell>

■ Сопроцессор Intel Xeon Phi 3120A

- **57 ядер** с микроархитектурой Intel Knights Corner: in-order, 4-way SMT,
SIMD Unit: 1 векторная AVX-512 операция за такт, 32 векторных регистра,
vector gather/scatter, IEEE 754 2008
- **Память:** 6 GiB
- **Программное обеспечение:** MPSS 3.8



<http://semiaccurate.com/2012/08/28/intel-details-knights-corner-architecture-at-long-last/>

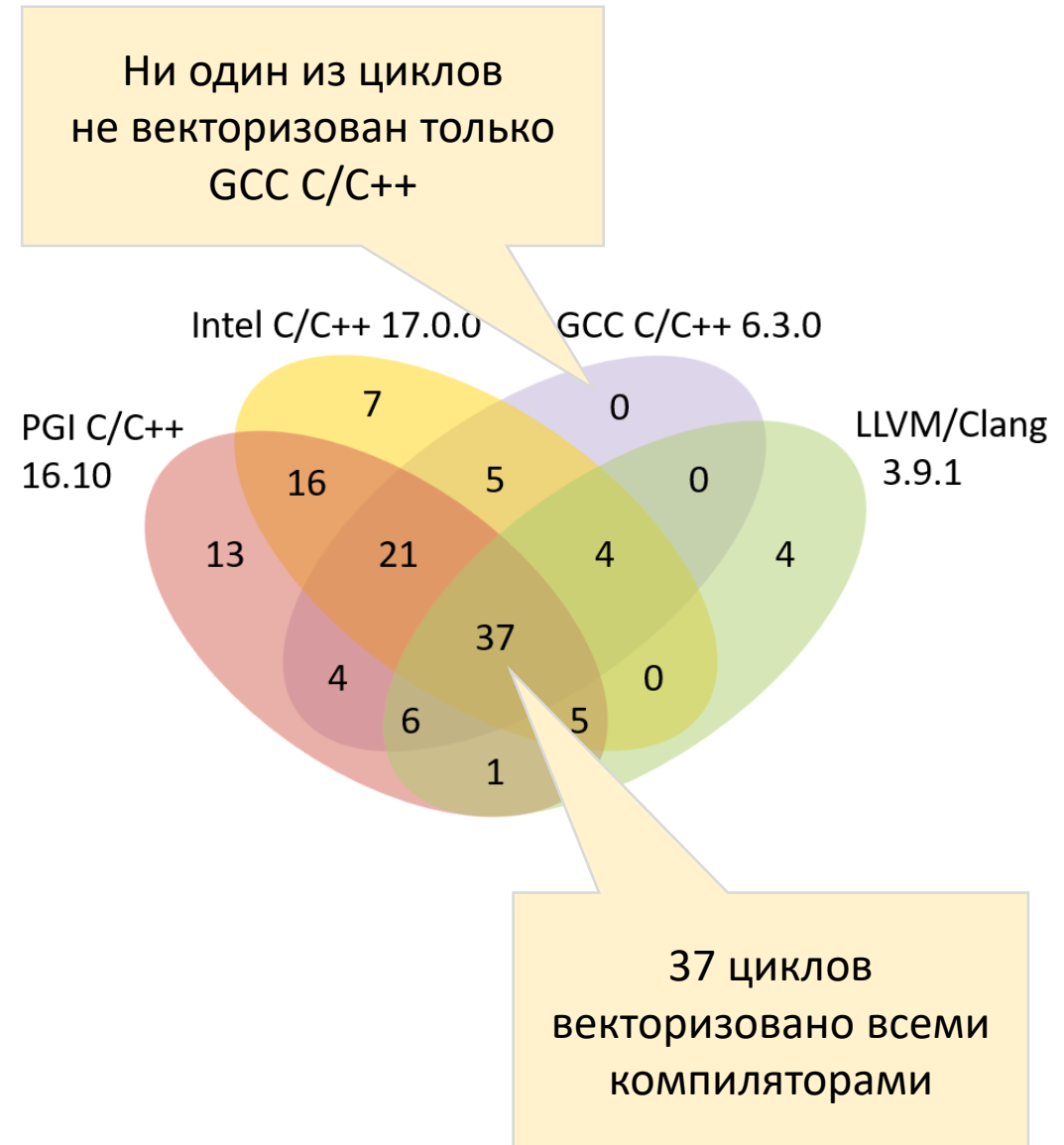
Целевые компиляторы

Компилятор	Опции компиляции	Отключение векторизатора
Intel C/C++ 17.0	<code>-O3 -xHost -qopt-report3 -qopt-report-phase=vec,loop -qopt-report-embed</code>	<code>-no-vec</code>
GCC C/C++ 6.3.0	<code>-O3 -ffast-math -fivopts -march=native -fopt-info-vec -fopt-info-vec-missed -fno-tree-vectorize</code>	<code>-fno-tree-vectorize</code>
LLVM/Clang 3.9.1	<code>-O3 -ffast-math -fvectorize -Rpass=loop-vectorize -Rpass-missed=loop-vectorize -Rpass-analysis=loop-vectorize</code>	<code>-fno-vectorize</code>
PGI C/C++ 16.10 Community Edition	<code>-O3 -Mvect -Minfo=loop,vect -Mneginfo=loop,vect</code>	<code>-Mnovect</code>

Результаты экспериментов

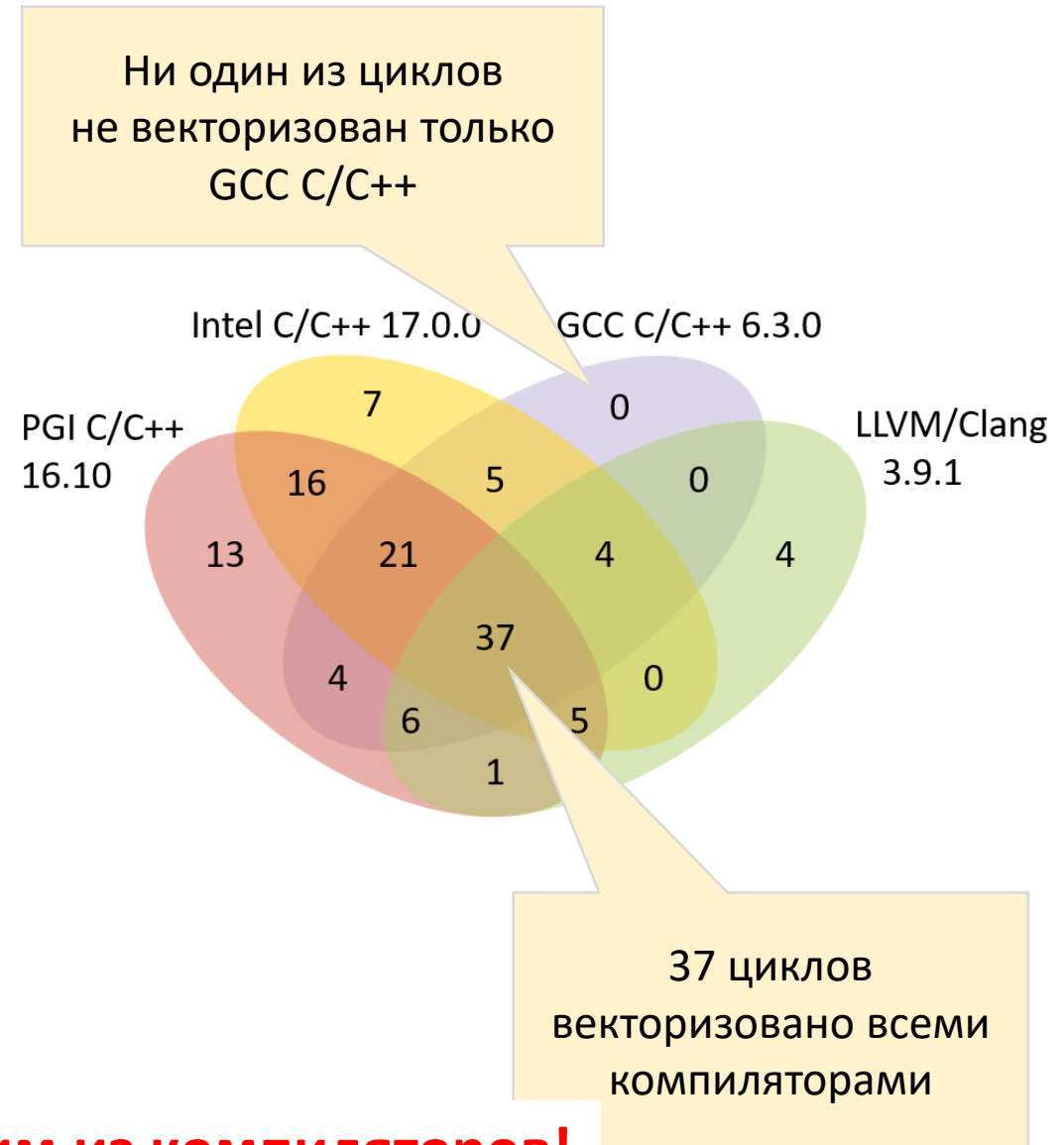
Количество автоматически векторизованных циклов

Компиляторы	Число циклов, векторизованных <u>только</u> указанными компиляторами
Intel C/C++, GCC C/C++, LLVM/Clang, PGI C/C++	37
Intel C/C++, GCC C/C++, LLVM/Clang	4
Intel C/C++, GCC C/C++, PGI C/C++	21
Intel C/C++, LLVM/Clang, PGI C/C++	5
GCC C/C++, LLVM/Clang, PGI C/C++	6
Intel C/C++, GCC C/C++	5
Intel C/C++, LLVM/Clang	0
Intel C/C++, PGI C/C++	16
GCC C/C++, PGI C/C++	4
GCC C/C++, LLVM/Clang	0
LLVM/Clang, PGI C/C++	1
Intel C/C++	7
GCC C/C++	0
LLVM/Clang	4
PGI C/C++	13



Количество автоматически векторизованных циклов

Компиляторы	Число циклов, векторизованных <u>только</u> указанными компиляторами
Intel C/C++, GCC C/C++, LLVM/Clang, PGI C/C++	37
Intel C/C++, GCC C/C++, LLVM/Clang	4
Intel C/C++, GCC C/C++, PGI C/C++	21
Intel C/C++, LLVM/Clang, PGI C/C++	5
GCC C/C++, LLVM/Clang, PGI C/C++	6
Intel C/C++, GCC C/C++	5
Intel C/C++, LLVM/Clang	0
Intel C/C++, PGI C/C++	16
GCC C/C++, PGI C/C++	4
GCC C/C++, LLVM/Clang	0
LLVM/Clang, PGI C/C++	1
Intel C/C++	7
GCC C/C++	0
LLVM/Clang	4



PGI

28 циклов (18,5 %) не векторизованы ни одним из компиляторов!

Результаты автоматической векторизации циклов (Intel 64, тип данных double)

V	Цикл векторизован полностью	M	Мультиверсионность	NI	Невозможно вычислить количество итераций	OL	Значение не может быть использовано за пределами цикла
PV	Цикл векторизован частично	BO	Неподходящая операция	CF	Невозможно определить направление потока управления	UV	Векторизатор не может понять поток управления в цикле
RV	Остаток цикла не векторизован	AP	Сложный шаблон доступа к элементам массива	SS	Цикл не подходит для векторной записи по несмежным адресам	SW	Наличие оператора switch в цикле
IF	Векторизация возможна, но не эффективна	R	Значение, которое не может быть идентифицировано как результат редукции, используется вне цикла	ME	Цикл с несколькими выходами невозможно векторизовать	US	Неподдерживаемое использование в выражении
D	Зависимость по данным препятствует векторизации	IL	Переменная-счетчик внутреннего цикла не является инвариантом	FC	Цикл содержит вызовы функций или данные, которые невозможно проанализировать	GS	В базовом блоке нет сгруппированных операций записи

Цикл S235 векторизован GCC и PGI

ICC	V	IF	V	IF	V	V	D	V	V	V	V	D	V	D	V	D	V	V	V	V	D	V	D	D	V	V	D	V	V	V	V	PV	V	PV	M	D	IF	V	V	PV	PV	PV								
PGI	V	IF	V	V	V	V	D	V	V	V	D	V	V	V	NI	D	D	V	D	V	D	V	D	D	NI	V	V	NI	V	D	D	D	D	D	V	D	IF	V	V	V	D	D								
LLVM	V	IF	V	D	V	D	D	D	IF	R	R	V	V	V	NI	CF	CF	V	R	IF	IF	V	V	R	V	V	CF	CF	V	V	NI	V	V	NI	V	R	R	R	R	V	R	R	R	IF	R	R	R	D	R	
GCC	V	V	V	V	V	V	V	BO	V	AP	D	IL	V	V	V	NI	SS	V	V	AP	V	V	V	V	IL	V	M	M	M	M	V	NI	V	V	NI	V	D	D	D	D	V	D	V	IL	IL	D	M	V	D	D

Цикл:

	S000	S111	S1111	S112	S1112	S113	S1113	S114	S115	S1115	S116	S118	S119	S1119	S121	S122	S123	S124	S125	S126	S127	S128	S131	S132	S141	S151	S152	S161	S1161	S162	S171	S172	S173	S174	S175	S176	S211	S212	S1213	S221	S1221	S222	S231	S232	S1232	S233	S235	S241	S242				
ICC	PV	PV	PV	V	V	V	D	V	V	V	V	D	D	D	D	V	V	V	V	V	V	PV	V	D	V	V	V	V	V	V	V	D	V	V	V	D	IF	PV	D	V	IF	V	V	V	IF	V	V	V	V	D	V		
PGI	D	D	D	D	V	V	V	V	V	V	V	V	D	D	V	D	V	V	V	V	D	V	IF	D	V	V	V	V	V	V	D	V	V	V	D	IF	V	V	V	FC	V	V	V	V	V	V	V	V	V	V	V		
LLVM	V	D	D	V	V	V	R	V	V	CF	V	R	R	OL	R	R	CF	CF	CF	CF	R	IF	UV	CF	CF	CF	CF	CF	CF	CF	D	V	R	R	D	IF	IF	R	V	FC	V	V	V	R	V	V	R	V	R	V			
GCC	D	D	D	V	V	V	US	V	US	V	FC	US	AP	BO	US	D	V	V	V	V	CF	V	V	D	M	CF	CF	V	V	V	V	V	US	US	V	IF	IF	D	V	FC	V	V	V	CF	V	V	US	V	V	V			
	S243	S244	S1244	S2244	S251	S1251	S2251	S3251	S252	S253	S254	S255	S256	S257	S258	S261	S271	S272	S273	S274	S275	S2275	S276	S277	S278	S279	S1279	S2710	S2711	S2712	S281	S1281	S291	S292	S293	S2101	S2102	S2111	S311	S31111	S312	S313	S314	S315	S316	S317	S318	S319	S3110	S13110			
ICC	V	D	V	D	D	D	V	ME	D	D	D	V	V	V	IF	V	V	M	M	M	V	V	V	V	V	V	V	V	ME	ME	IF	V	IF	V	V	V	V	V	V	V	IF	V	V	V	V	V	V	V	V	V	V	V	V
PGI	V	V	V	D	D	D	V	ME	V	V	V	V	V	V	IF	V	V	V	V	V	V	V	V	V	V	V	IF	V	V	FC	ME	V	V	IF	D	V	V	V	V	V	IF	V	V	V	V	V	V	V	V	V	V	V	
LLVM	R	R	V	R	R	R	R	UV	CF	CF	CF	IF	V	IF	IF	V	V	V	V	V	V	CF	SW	CF	V	V	R	V	UV	NI	IF	IF	IF	IF	IF	IF	IF	V	FC	IF	IF	CF	V	V	V	V	V	V	V	V	V		
GCC	V	US	V	D	D	D	US	CF	SS	US	SS	V	V	GS	BO	V	V	V	V	V	V	CF	CF	V	BO	V	V	V	CF	CF	SS	V	SS	V	V	V	V	V	V	CF	V	SS	V	V	V	V	V	V	V	V	V	V	
	S3111	S3112	S3113	S321	S322	S323	S331	S332	S341	S342	S343	S351	S1351	S352	S353	S421	S1421	S422	S423	S424	S431	S441	S442	S443	S451	S452	S453	S461	S481	S482	S491	S4112	S4113	S4114	S4115	S4116	S4117	S4121	va	vag	vas	vif	vpv	vrv	vpvrv	vpvrs	vpvrv	vsumr	vdotr	vbor			

Результаты автоматической векторизации циклов (Intel Xeon Phi, тип данных double)

V	Цикл векторизован полностью	M	Мультиверсионность	NI	Невозможно вычислить количество итераций	OL	Значение не может быть использовано за пределами цикла
PV	Цикл векторизован частично	BO	Неподходящая операция	CF	Невозможно определить направление потока управления	UV	Векторизатор не может понять поток управления в цикле
RV	Остаток цикла не векторизован	AP	Сложный шаблон доступа к элементам массива	SS	Цикл не подходит для векторной записи по несмежным адресам	SW	Наличие оператора switch в цикле
IF	Векторизация возможна, но не эффективна	R	Значение, которое не может быть идентифицировано как результат редукции, используется вне цикла	ME	Цикл с несколькими выходами невозможно векторизовать	US	Неподдерживаемое использование в выражении
D	Зависимость по данным препятствует векторизации	IL	Переменная-счетчик внутреннего цикла не является инвариантом	FC	Цикл содержит вызовы функций или данные, которые невозможно проанализировать	GS	В базовом блоке нет сгруппированных операций записи

Intel C/C++ Compiler 17.0 (-mmic) native mode

double	V	V	V	V	V	V	D	V	V	V	PV	D	V	D	V	D	D	V	V	D	V	V	V	D	V	D	D	V	M	D	D	V	V	D	V	V	V	V	PV	RV	PV	M	D	IF	V	V	PV	PV	D				
float	V	V	V	V	V	V	D	V	V	V	PV	D	V	D	V	D	D	V	V	D	V	V	V	D	V	D	D	V	M	D	D	V	V	D	V	V	V	V	PV	PV	PV	M	D	V	V	V	PV	PV	PV				
	S000	S111	S1111	S112	S1112	S113	S1113	S114	S115	S1115	S116	S118	S119	S1119	S121	S122	S123	S124	S125	S126	S127	S128	S131	S132	S141	S151	S152	S161	S1161	S162	S171	S172	S173	S174	S175	S176	S211	S212	S1213	S221	S1221	S222	S231	S232	S1232	S233	S2333	S235	S241	S242			
double	PV	PV	PV	V	V	V	D	V	V	V	V	D	D	D	D	V	V	V	V	V	V	PV	V	D	V	V	V	V	V	V	D	V	V	V	D	V	PV	D	V	IF	V	V	V	V	V	V	V	V	D	V			
float	PV	PV	PV	V	V	V	D	V	V	V	V	D	D	D	D	V	V	V	V	V	V	PV	V	D	V	V	V	V	V	V	D	V	V	V	D	V	PV	D	V	IF	V	V	V	V	V	V	V	V	D	V			
	S243	S244	S1244	S2244	S251	S1251	S2251	S3251	S252	S253	S254	S255	S256	S257	S258	S261	S271	S272	S273	S274	S275	S2275	S276	S277	S278	S279	S1279	S2710	S2711	S2712	S281	S1281	S291	S292	S293	S2101	S2102	S2111	S311	S31111	S312	S313	S314	S315	S316	S317	S318	S319	S3110	S13110			
double	V	D	V	D	D	D	V	ME	V	V	V	V	PV	V	V	V	V	V	M	M	M	V	V	V	V	V	V	V	ME	ME	V	V	V	PV	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
float	V	D	V	D	D	D	V	ME	V	V	V	V	V	V	V	V	V	M	M	M	V	V	V	V	V	V	V	V	ME	ME	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
	S3111	S3112	S3113	S321	S322	S323	S331	S332	S341	S342	S343	S351	S1351	S352	S353	S421	S1421	S422	S423	S424	S431	S441	S442	S443	S451	S452	S453	S461	S481	S482	S491	S4412	S4413	S4414	S4415	S4416	S4417	S4421	va	vag	vas	vif	vpv	vtv	vpvtv	vpvts	vpvpv	vtvtv	vsumr	vdotr	vbor		

Классы не векторизованных циклов

Классы не векторизованных циклов

Категория / Подкатегория	Общее число циклов	Число не векторизованных циклов
Анализ зависимостей по данным (dependence analysis)	36	9
Линейная зависимость по данным (linear dependence)	14	2
Распознавание индуктивной переменной (induction variable recognition)	8	3
Нелинейная зависимость (nonlinear dependence)	1	1
Условные и безусловные переходы (control flow)	3	1
Переменные в границах цикла или шаге выполнения итераций (symbolics)	6	2
Анализ потока управления и трансформация циклов (vectorization)	52	11
Расщепление тела цикла (loop distribution)	3	2
Перестановка циклов (loop interchange)	6	2
Расщепление вершин в графе зависимостей по данным (node splitting)	6	4
Растягивание скаляров и массивов» (scalar and array expansion)	12	2
Условные и безусловные переходы (control flow)	14	1
Распознавание идиоматических конструкций (idiom recognition)	27	6
Рекуррентности (recurrences)	3	3
Поиск элемента в массиве (search loops)	2	1
Свертка цикла (loop rerolling)	4	1
Редукции (reductions)	15	1
Полнота понимания языка программирования (language completeness)	23	2
Прерывание вычислений в цикле» (nonlocal GOTO)	2	2

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Линейная зависимость по данным» (linear dependence)

Невекторизованный цикл s1113

```
for (int i = 0; i < N; i++)  
    X[i] = X[N/2] + Y[i];
```

- Зависимость по данным вида «чтение после записи» (read-after-write, RAW), начиная с итерации $N / 2 + 1$

Раскрытие цикла s1113 по итерациям

$X[0] = X[N/2] + Y[0];$

$X[1] = X[N/2] + Y[1];$

...

$X[N/2] = X[N/2] + Y[N/2];$

$X[N/2+1] = X[N/2] + Y[N/2+1];$

...

$X[N-1] = X[N/2] + Y[N-1];$

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Линейная зависимость по данным» (linear dependence)

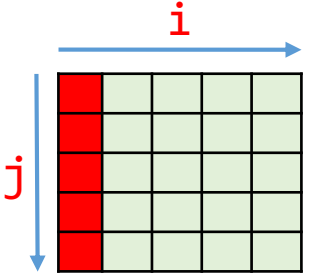
Невекторизованный цикл s1113	Возможная трансформация цикла s1113
<pre>for (int i = 0; i < N; i++) X[i] = X[N/2] + Y[i];</pre>	<pre>int k = N / 2; for (int i = 0; i <= k; i++) X[i] = X[k] + Y[i]; for (int i = k + 1; i < N; i++) X[i] = X[k] + Y[i];</pre>

- Зависимость по данным вида «чтение после записи» (read-after-write, RAW), начиная с итерации $N / 2 + 1$

- *Распределение путем разбиения имен* (fission by name)
- Ускорение в 2 раза для типа `double` и компилятора ICC на Intel Xeon E5-2620 v4

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Распознавание индуктивной переменной» (induction variable recognition)

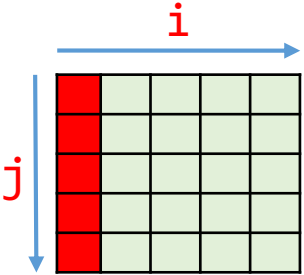
Невекторизованный цикл s126	Возможная трансформация цикла s126
<pre>int k = 1; for (int i = 0; i < N; i++) { for (int j = 1; j < N; j++) { X[j][i] = X[j - 1][i] + Y[k - 1] * Z[j][i]; ++k; } ++k; }</pre>  <p style="text-align: center;">Шаг = N</p>	<pre>for (int j = 1; j < N; j++) { for (int i = 0; i < N; i++) { X[j][i] = X[j - 1][i] + Y[i * 4 + j - 1] * Z[j][i]; } }</pre>

- Индуктивная переменная k
- Внешний цикл осуществляет проход по столбцам матриц X и Z , а внутренний – по строкам

- **Перестановка циклов и удаление индуктивной переменной k**
- Ускорение в 8.5 раз для типа `double` и компилятора ICC на Intel Xeon E5-2620 v4

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Нелинейная зависимость» (nonlinear dependence)

Невекторизованный цикл s141	Возможная трансформация цикла s141
<pre>int k; for (int i = 0; i < N; i++) { k = (i + 1) * ((i + 1) - 1) / 2 + (i + 1) - 1; for (int j = i; j < N; j++) { X[k] += Y[j][i]; k += j + 1; } }</pre>  <p style="text-align: center;">Шаг = N</p>	<pre>int k = 0; for (int j = 0; j < N; j++) { for (int i = 0; i <= j; i++) { X[k] += Y[j][i]; k++; } }</pre>
<ul style="list-style-type: none">▪ Значение индуктивной переменной k зависит от i и j▪ Внешний цикл осуществляет проход по столбцам матрицы Y, а внутренний – по строкам	<ul style="list-style-type: none">▪ Упрощение вычисления значения индуктивной переменной k и перестановка циклов▪ Ускорение в 3.5 раза для типа <code>double</code> и компилятора ICC на Intel Xeon E5-2620 v4

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Условные и безусловные переходы» (control flow)

Невекторизованный цикл s161

```
for (int i = 0; i < N - 1; ++i)
{
    if (Y[i] < 0)
        goto L20;

    X[i] = Z[i] + V[i] * W[i];
    goto L10;

L20: Z[i + 1] = X[i] + V[i] * V[i];
L10: ;
}
```

Возможная зависимость выражения S_2 на итерации i от
выражения S_1 на итерации $i - 1$
вида «чтение после записи» (read-after-write, RAW)

Раскрытие цикла s161 по итерациям

```
i = 0:
    if (Y[0] < 0)
S10:     Z[1] = X[0] + V[0] * V[0];
    else
S20:     X[0] = Z[0] + V[0] * W[0];

i = 1:
    if (Y[1] < 0)
S11:     Z[2] = X[1] + V[1] * V[1];
    else
S21:     X[1] = Z[1] + V[1] * W[1];
```

$S_1^0 \delta < S_2^1$

Категория «Анализ зависимостей по данным» (dependence analysis)

Подкатегория «Переменные в границах цикла или шаге выполнения итераций»
(symbolics)

Невекторизованный цикл s172

```
void s172(int n1, int n3)
{
    for (int i = n1 - 1; i < N; i += n3)
        X[i] += Y[i];
}
```

Переменные, используемые в качестве нижней и(или)
верхней границы цикла и(или) шага выполнения
итераций

Категория «Анализ потока управления и трансформация циклов» (vectorization)

Подкатегория «Расщепление тела цикла» (loop distribution)

Невекторизованный цикл s221

```
for (int i = 1; i < N; i++)  
{  
  S1:    X[i] += Z[i] * V[i];           S1 δ= S2  
  S2:    Y[i] = Y[i-1] + X[i] + V[i];   S2 δ< S2  
}
```

- Зависимость выражения S_2 на итерации i от выражения S_1 на той же итерации вида «чтение после записи» (read-after-write, RAW)
- Зависимость выражения S_2 на итерации i от выражения S_2 на итерации $i - 1$ вида «чтение после записи» (read-after-write, RAW)

Категория «Анализ потока управления и трансформация циклов» (vectorization)

Подкатегория «Перестановка циклов» (loop interchange)

Невекторизованный цикл s1232

```
for (int j = 0; j < N; j++)  
  for (int i = j; i < N; i++)  
    X[i][j] = Y[i][j] + Z[i][j];
```

- Внешний цикл осуществляет проход по столбцам матриц X, Y, Z, а внутренний – по строкам

Возможная трансформация цикла s1232

```
for (int i = 0; i < N; i++)  
  for (int j = 0; j <= i; j++)  
    X[i][j] = Y[i][j] + Z[i][j];
```

- **Перестановка циклов**
- Ускорение в 8.6 раз для типа double и компилятора ICC на Intel Xeon E5-2620 v4

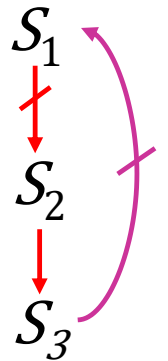
Категория «Анализ потока управления и трансформация циклов» (vectorization)

Подкатегория «Расщепление вершин в графе зависимостей по данным» (node splitting)

Невекторизованный цикл s244

```
for (int i = 0; i < N - 1; ++i)
{
  S1:   X[i] = Y[i] + Z[i] * V[i];
  S2:   Y[i] = Z[i] + Y[i];
  S3:   X[i + 1] = Y[i] + X[i + 1] * V[i];
}
```

$S_1 \bar{\delta} = S_2$
 $S_2 \delta = S_3$
 $S_1 \bar{\delta} < S_3$



- Зависимость выражения S_2 на итерации i от выражения S_1 на той же итерации вида «запись после чтения» (write-after-read, WAR)
- Зависимость выражения S_3 на итерации i от выражения S_2 на той же итерации вида «чтение после записи» (read-after-write, RAW)
- Зависимость выражения S_1 на итерации i от выражения S_3 на итерации $i - 1$ вида «запись после чтения» (write-after-read, WAR)

Категория «Анализ потока управления и трансформация циклов» (vectorization)

Подкатегория «Растягивание скаляров и массивов» (scalar and array expansion)

Невекторизованный цикл s257	Возможная трансформация цикла s257
<pre> for (int i = 1; i < N; i++) { for (int j = 0; j < N; j++) { S₁: X[i] = Y[j][i] - X[i - 1]; S₂: Y[j][i] = X[i] + Z[j][i]; } } </pre> <p style="text-align: center;">$S_1 \delta_{<} S_1$ $S_1 \delta_{=} S_2$ $S_1 \bar{\delta}_{=,=} S_2$</p>	<pre> for (int i = 1; i < N; i++) for (int j = 0; j < N; j++) X[i] = Y[j][i] - X[i - 1]; for (int i = 1; i < N; i++) for (int j = 0; j < N; j++) Y[j][i] = Y[j][i] - X[i - 1] + Z[j][i]; </pre>

- Зависимость выражения S_1 на итерации i от S_1 на итерации $i - 1$ вида «чтение после записи» (read-after-write)
- Зависимость выражения S_2 на итерации i от S_1 на той же итерации вида «чтение после записи» (read-after-write)
- Зависимость выражения S_2 на итерациях i, j от S_1 на тех же итерациях вида «запись после чтения» (write-after-read)

- **Расщепление тела цикла** (loop fission, loop distribution)
- Ускорение в 9.3 раза для типа double и компилятора ICC на Intel Xeon E5-2620 v4

Категория «Распознавание идиоматических конструкций» (idiom recognition)

Подкатегория «Рекуррентности» (recurrences)

Невекторизованный цикл s322

```
 $S_1$ :   for (int i = 2; i < N; i++)  
        {  
             $S_1$     $X[i] = X[i] + X[i-1] * Y[i] + X[i-2] * Z[i];$     $S_1 \delta < S_1$   
        }
```

Зависимость выражения S_1 на итерации i от выражения S_1 на итерациях $i-1$ и $i-2$ вида «чтение после записи» (read-after-write, RAW)

Категория «Распознавание идиоматических конструкций» (idiom recognition)

Подкатегория «Поиск элемента в массиве» (search loops)

Невекторизованный цикл s332	Возможная трансформация цикла s332
<pre>void s332(double t) { int index = -2; double value = (double)-1.; for (int i = 0; i < N; i++) { if (X[i] > t) { index = i; value = X[i]; goto L20; } } L20: }</pre>	<pre>void s332(double t) { int index = -2; double value = (double)-1.; for (i = 0; i < N && X[i] <= t; i++) ; if (X[i] > t) { index = i; value = X[i]; } }</pre> <p><i>Векторизован только компилятором Intel C/C++</i></p>

- Условный и безусловный переходы в теле цикла

- Условный переход вынесен за пределы тела цикла
- Ускорение в 2.4 раза для типа double и компилятора ICC на Intel Xeon E5-2620 v4

Категория «Распознавание идиоматических конструкций» (idiom recognition)

Подкатегория «Свертка цикла» (loop rerolling)

Невекторизованный цикл s353

```
void s353(int* __restrict__ ip) {  
    double alpha = Z[0];  
    for (int i = 0; i < N; i += 5) {  
        X[i]    += alpha * Y[ip[i]];  
        X[i+1] += alpha * Y[ip[i+1]];  
        X[i+2] += alpha * Y[ip[i+2]];  
        X[i+3] += alpha * Y[ip[i+3]];  
        X[i+4] += alpha * Y[ip[i+4]];  
    }  
}
```

Косвенная адресация при обращении к элементам массива $Y[ip[i]]$

Категория «Распознавание идиоматических конструкций» (idiom recognition)

Подкатегория «Редукции» (reductions)

Невекторизованный цикл s31111

```
double test(double* A) {
    double s = (double)0.;
    for (int i = 0; i < 4; i++)
        s += A[i];
    return s;
}

void s31111() {
    double sum;
    for (int i = 0; i < N; i++) {
        sum = (double)0.;
        sum += test(X);
        sum += test(&X[4]);
        sum += test(&X[8]);
        sum += test(&X[12]);
        sum += test(&X[16]);
        sum += test(&X[20]);
        sum += test(&X[24]);
        sum += test(&X[28]);
    }
}
```

Наличие вызова
функции в теле цикла

Категория «Полнота понимания языка программирования» (language completeness)

Подкатегория «Прерывание вычислений в цикле» (nonlocal GOTO)

Невекторизованный цикл s481

```
for (int i = 0; i < N; i++)  
{  
    if (V[i] < (double)0.)  
        exit(0);  
  
    X[i] += Y[i] * Z[i];  
}
```

Наличие вызова функции `exit` в теле цикла

Категория «Полнота понимания языка программирования» (language completeness)

Подкатегория «Прерывание вычислений в цикле» (nonlocal GOTO)

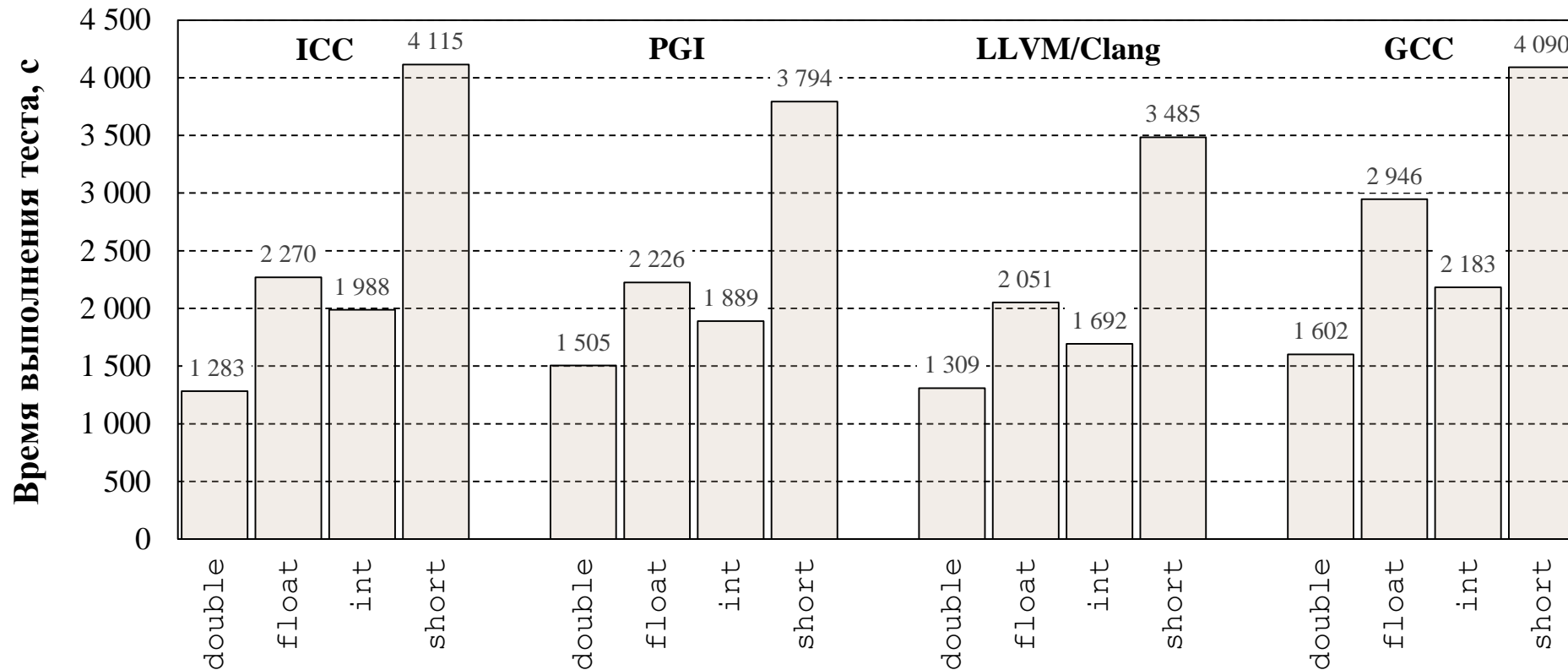
Невекторизованный цикл s482

```
for (int i = 0; i < N; i++)  
{  
    X[i] += Y[i] * Z[i];  
  
    if (Z[i] > Y[i])  
        break;  
}
```

Наличие break в теле цикла

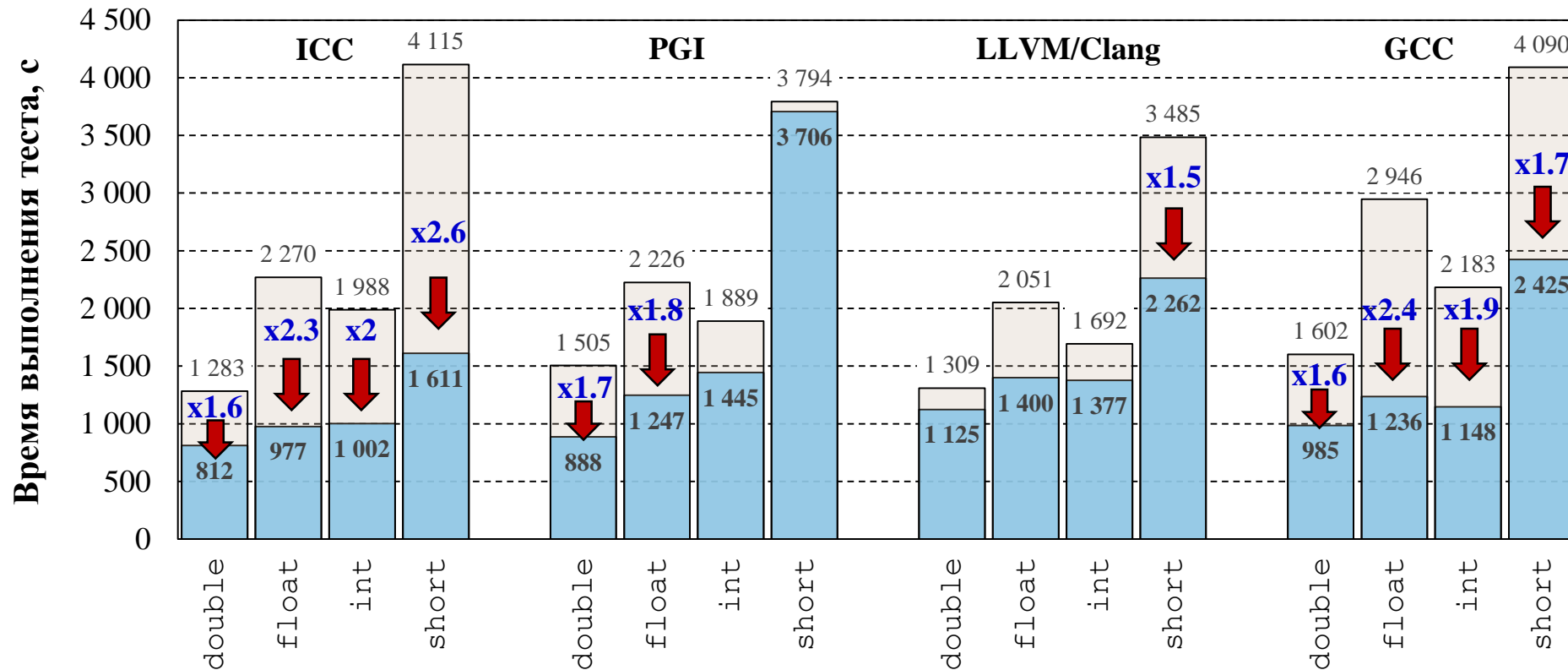
Сокращение времени выполнения векторизованных циклов

Время выполнения векторизованных циклов на процессоре Intel Xeon E5-2620 v4



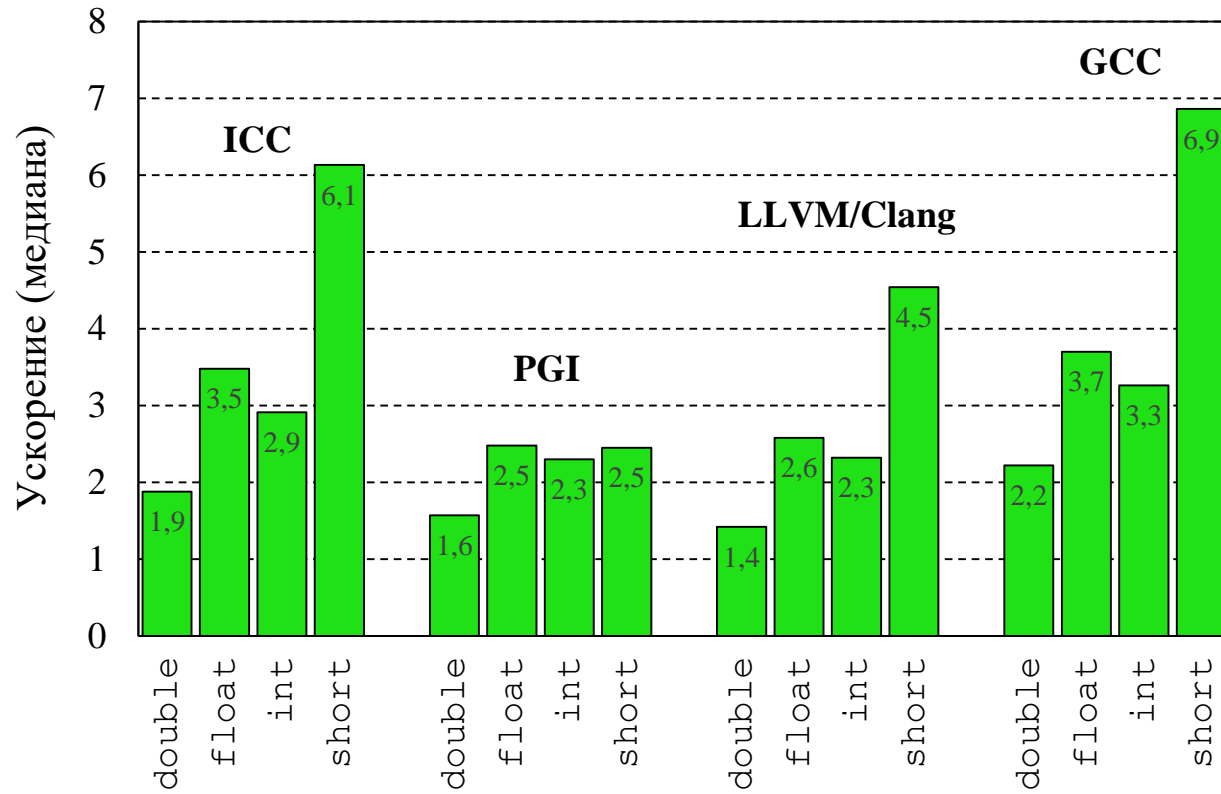
Суммарное время выполнения теста (всех 151 циклов)
для типов данных double, float, int и short int

Время выполнения векторизованных циклов на процессоре Intel Xeon E5-2620 v4

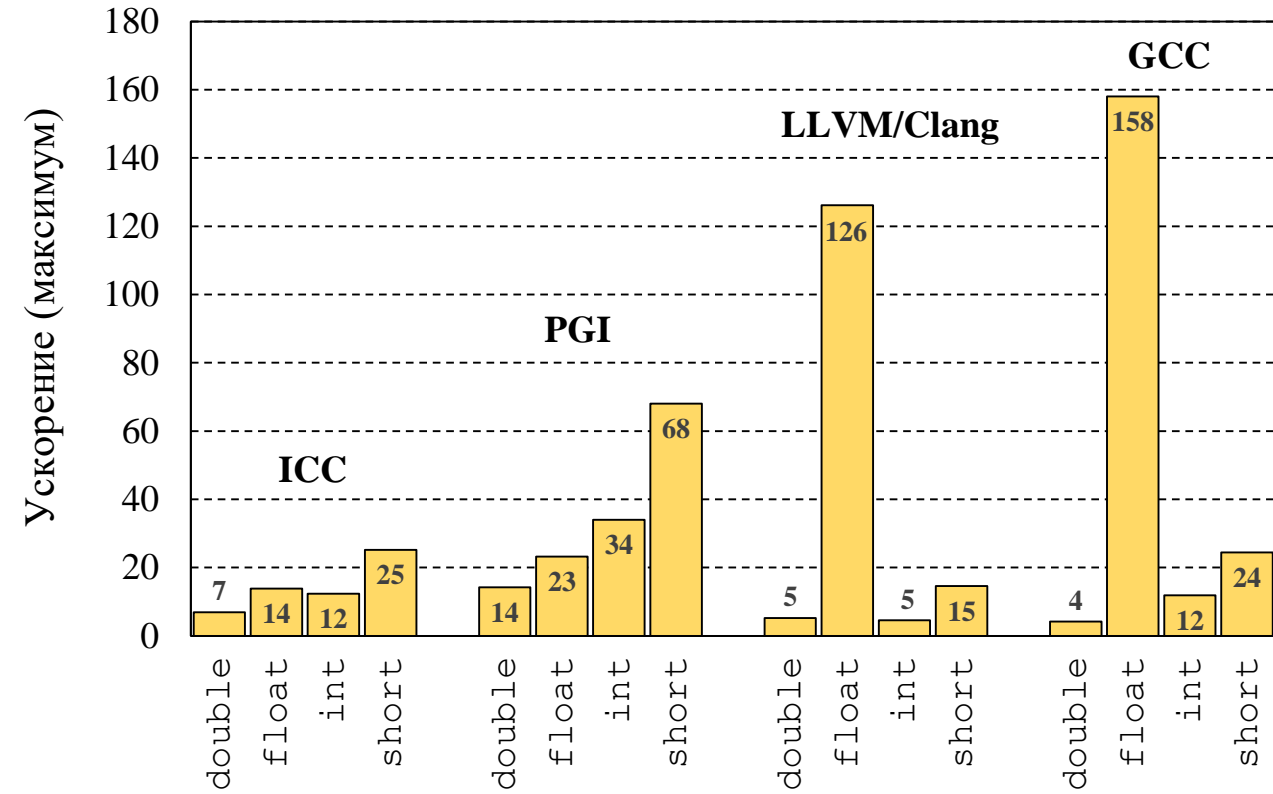


Суммарное время выполнения теста (всех 151 циклов)
для типов данных double, float, int и short int

Ускорение выполнения векторизованных циклов на процессоре Intel Xeon E5-2620 v4

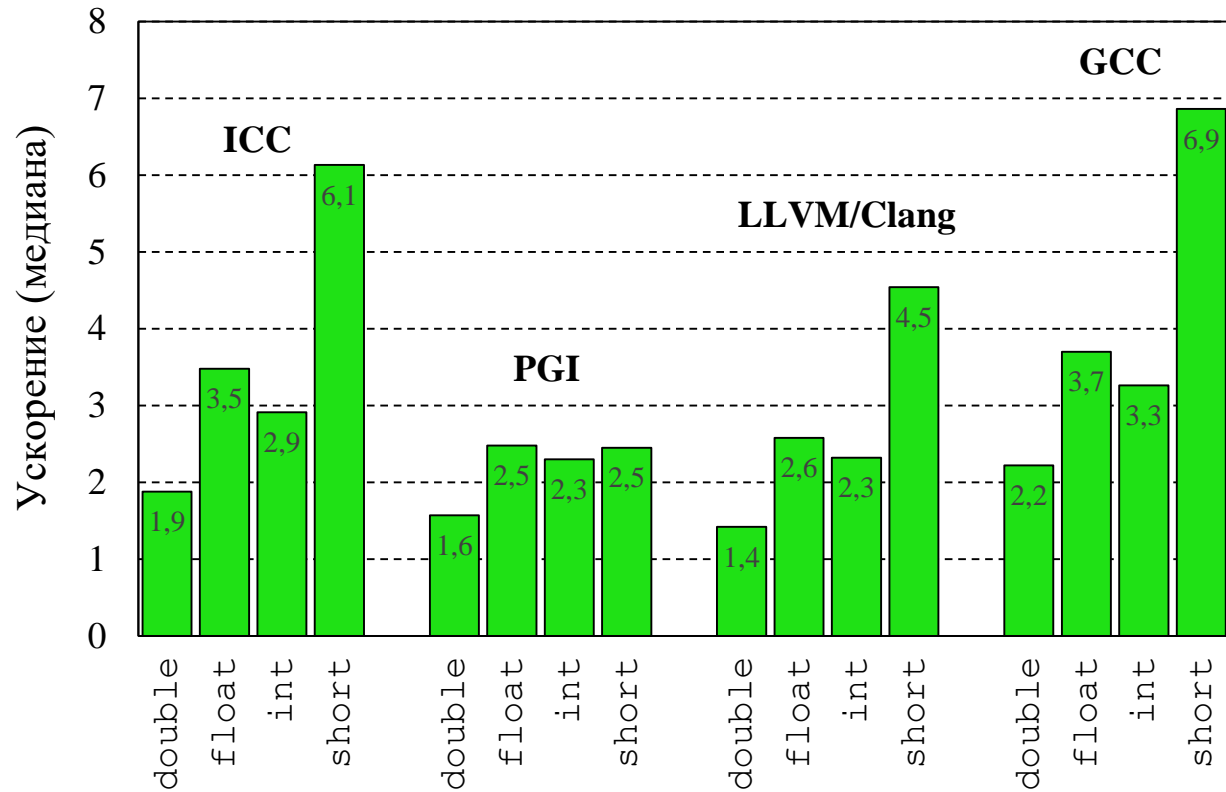


Медиана ускорения
(учитывались только ускорения больше 1.15)



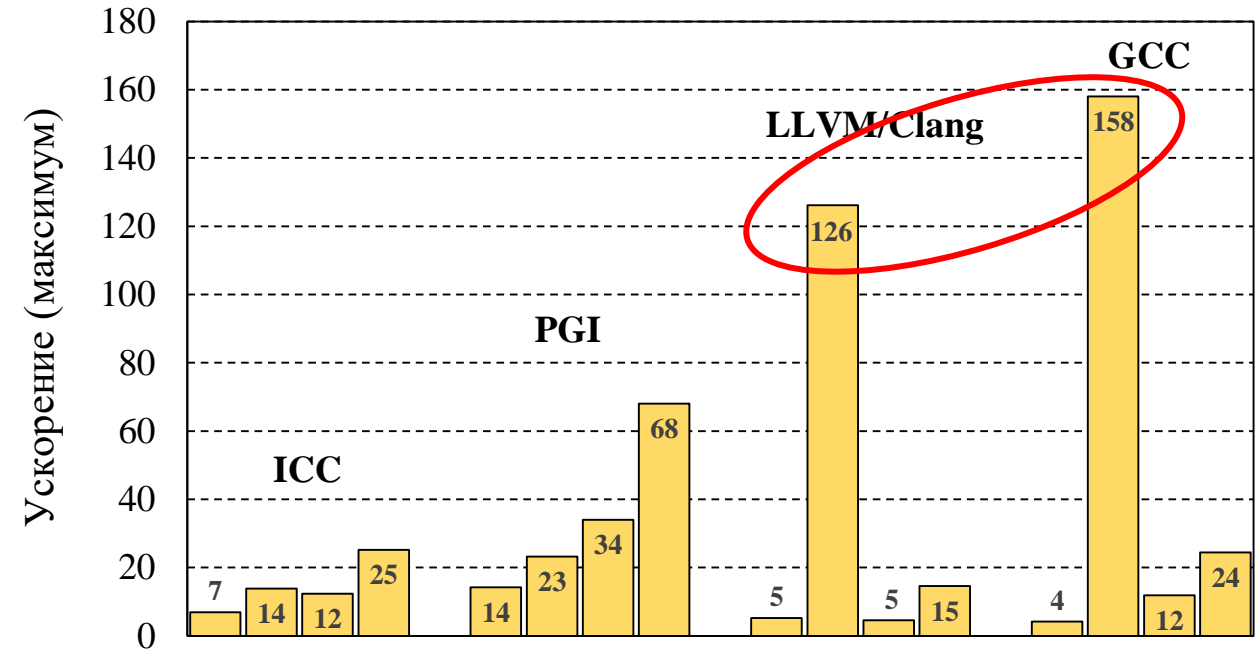
Максимальное ускорение

Ускорение выполнения векторизованных циклов на процессоре Intel Xeon E5-2620 v4



Медиана ускорения

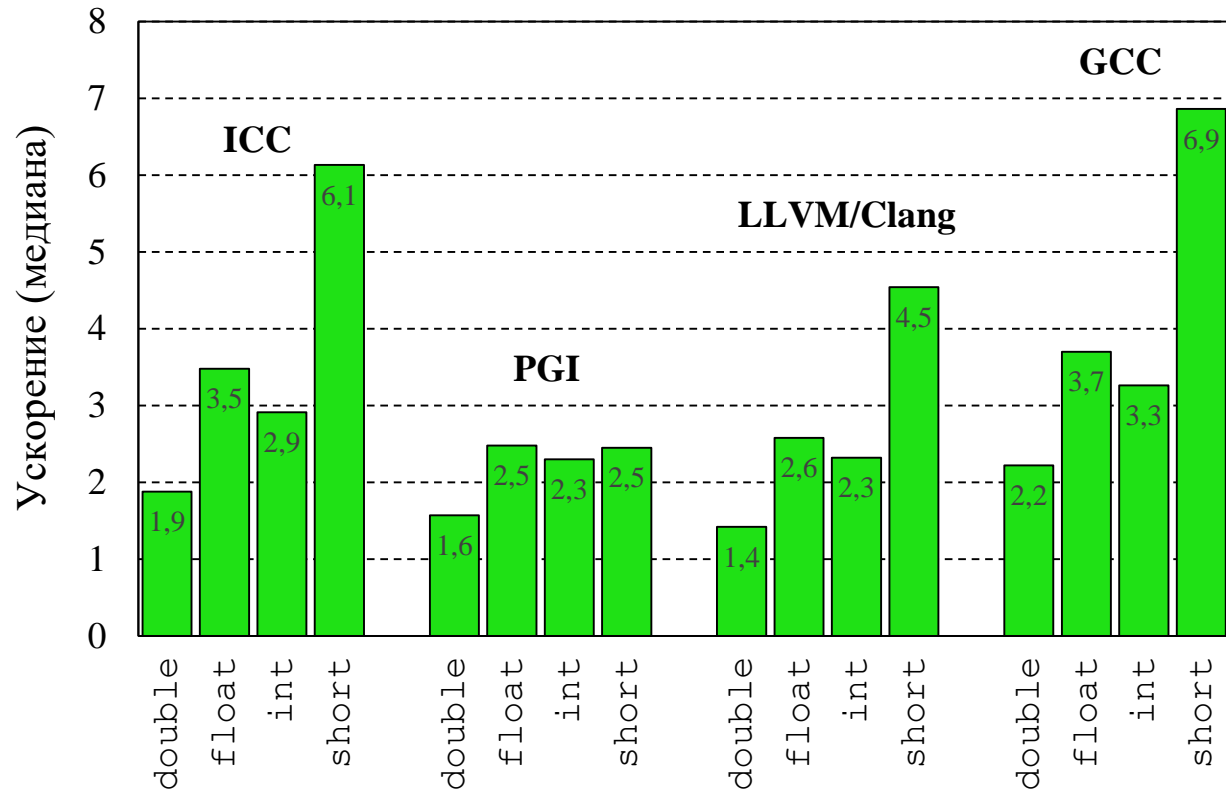
(учитывались только ускорения больше 1.15)



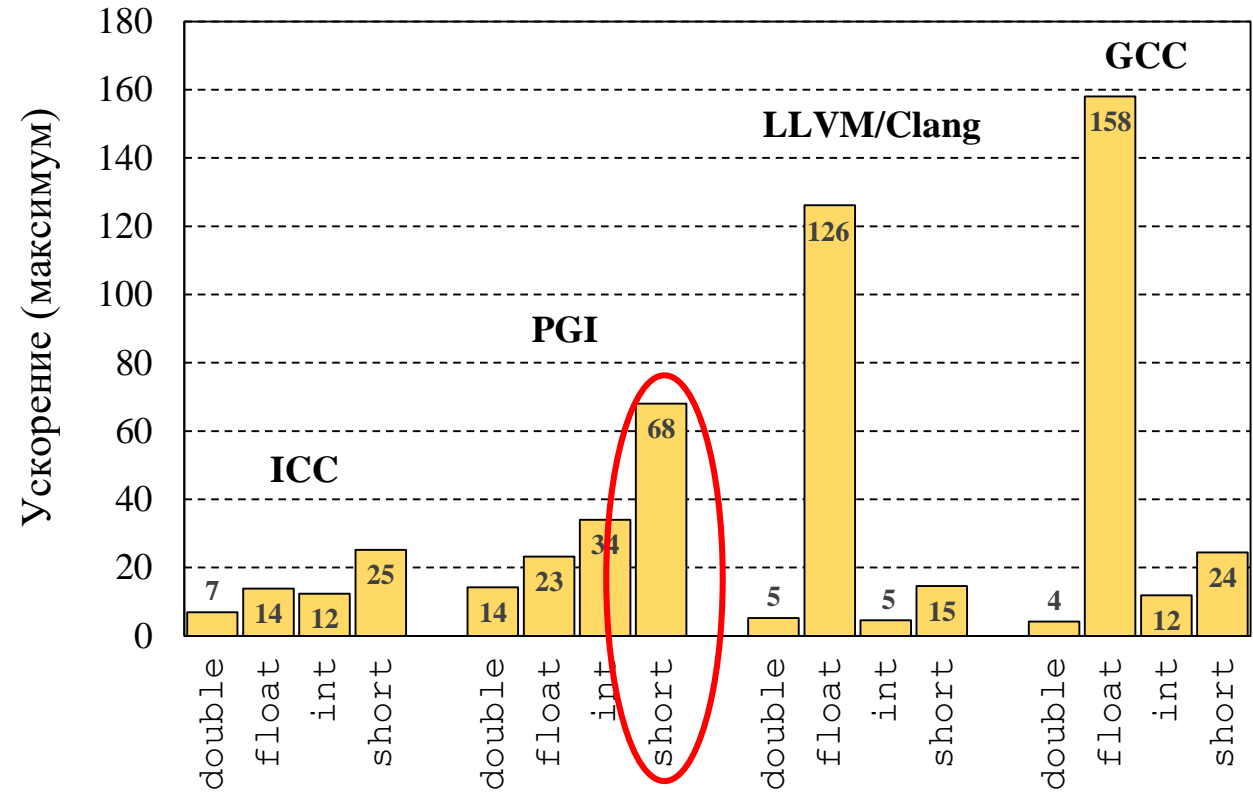
Эффективное распознавание операций чтения или записи по несмежным адресам памяти с использованием коэффициента чередования (interleaving factor) равного 2 [1]

[1] Nuzman D., Rosen I., Zaks A. *Auto-vectorization of interleaved data for SIMD* // Proc. of the 27th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI '06). 2006. pp. 132–143.

Ускорение выполнения векторизованных циклов на процессоре Intel Xeon E5-2620 v4



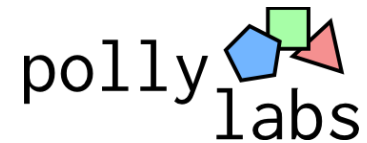
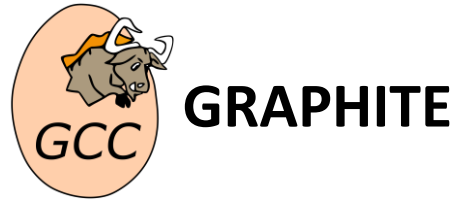
Медиана ускорения
(учитывались только ускорения больше 1.15)



Вычисления в цикле не производились в
результате оптимизации

Направление дальнейшей работы

- Анализ известных методов векторизации и распараллеливания циклов (полиэдральные модели: GCC Graphite, LLVM/Clang PollyLabs)



- Разработка методов векторизации установленного класса проблемных циклов из пакета ETSVC
- Анализ возможностей применения JIT-компиляции и оптимизации по результатам профилирования (profile-guided optimization) для автоматической векторизации кода

Спасибо за внимание!

Ольга Владимировна Молдованова ^{1,2}

ovm@sibguti.ru, ovm@isp.nsc.ru

Михаил Георгиевич Курносов ^{1,2}

WWW: www.mkurnosov.net

¹ Кафедра вычислительных систем

Сибирский государственный университет телекоммуникаций и информатики, Новосибирск

² Лаборатория вычислительных систем

Институт физики полупроводников им. А.В. Ржанова СО РАН, Новосибирск

Всероссийская научная конференция памяти А.Л. Фуксмана «Языки программирования и компиляторы» (PLC-2017)

г. Ростов-на-Дону, 3-5 апреля 2017 г.

Сравнение результатов с предыдущими работами

2011 г. [1]		2017 г.	
Intel C/C++ 12.0	90 циклов (59.6 %)	Intel C/C++ 17.0	95 циклов (62.9 %)
GCC C/C++ 4.7.0	59 циклов (39 %)	GCC C/C++ 6.3.0	79 циклов (52.3 %)

[1] Maleki S., Gao Ya. Garzarán M.J., Wong T., Padua D.A. *An Evaluation of Vectorizing Compilers* // Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT-11), 2011. pp. 372–382.

Ссылки на литературу

1. Maleki S., Gao Ya. Garzarán M.J., Wong T., Padua D.A. An Evaluation of Vectorizing Compilers // Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'11), 2011. pp. 372–382.
2. Extended Test Suite for Vectorizing Compilers. URL: <http://polaris.cs.uiuc.edu/~maleki1/TSVC.tar.gz>.
3. Callahan D., Dongarra J., Levine D. Vectorizing Compilers: A Test Suite and Results // Proc. of the ACM/IEEE conf. on Supercomputing (Supercomputing'88), 1988. pp. 98–105.
4. Levine D., Callahan D., Dongarra J. A Comparative Study of Automatic Vectorizing Compilers // Journal of Parallel Computing. 1991. Vol. 17. pp. 1223–1244.
5. Konsor P. Avoiding AVX-SSE Transition Penalties. URL: <https://software.intel.com/en-us/articles/avoiding-avx-sse-transition-penalties>.
6. Jibaja I., Jensen P., Hu N., Haghghat M., McCutchan J., Gohman D., Blackburn S., McKinley K. Vector Parallelism in JavaScript: Language and Compiler Support for SIMD // Proc. of the Int. Conf. on Parallel Architecture and Compilation (PACT-2015). 2015. pp. 407–418.
7. Векторизация программ: теория, методы, реализация. Сб. статей: Пер. с англ. и нем. М.: Мир, 1991. 275 с.
8. Metzger R.C., Wen Zh. Automatic Algorithm Recognition and Replacement: A New Approach to Program Optimization. MIT Press. 2000. 219 p.
9. Nuzman D., Rosen I., Zaks A. Auto-vectorization of interleaved data for SIMD // Proc. of the 27th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI '06). 2006. pp. 132–143.
10. Rohou E., Williams K., Yuste D. Vectorization Technology To Improve Interpreter Performance // ACM Transactions on Architecture and Code Optimization. 2013. 9 (4). pp. 26:1-26:22.

Виды зависимостей

- Потоковая (истинная) зависимость («чтение после записи», read-after-write, RAW)

```

for (int i = 0; i < N; i++)
{
S1:   A[i] = B[i] + C[i];
S2:   D[i] = A[i];
}
    
```

$$S_1 \delta = S_2$$



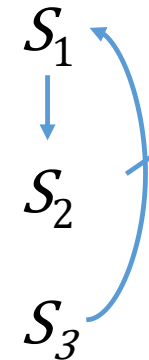
- Антизависимость («запись после чтения», write-after-read, WAR)

```

for (int i = 0; i < N; i++)
{
S1:   A[i] = B[i] + C[i];
S2:   D[i] = A[i];
S3:   B[i] = D[i];
}
    
```

$$S_1 \delta = S_2$$

$$S_3 \bar{\delta} = S_1$$



- Выходная зависимость («запись после записи», write-after-write, WAW)

```

for (int i = 0; i < N; i++)
{
S1:   A[i] = B[i] + C[i];
S2:   A[i+1] = A[i] + D[i];
}
    
```

$$S_1 \delta = S_2$$

$$S_2 \delta^o > S_1$$

