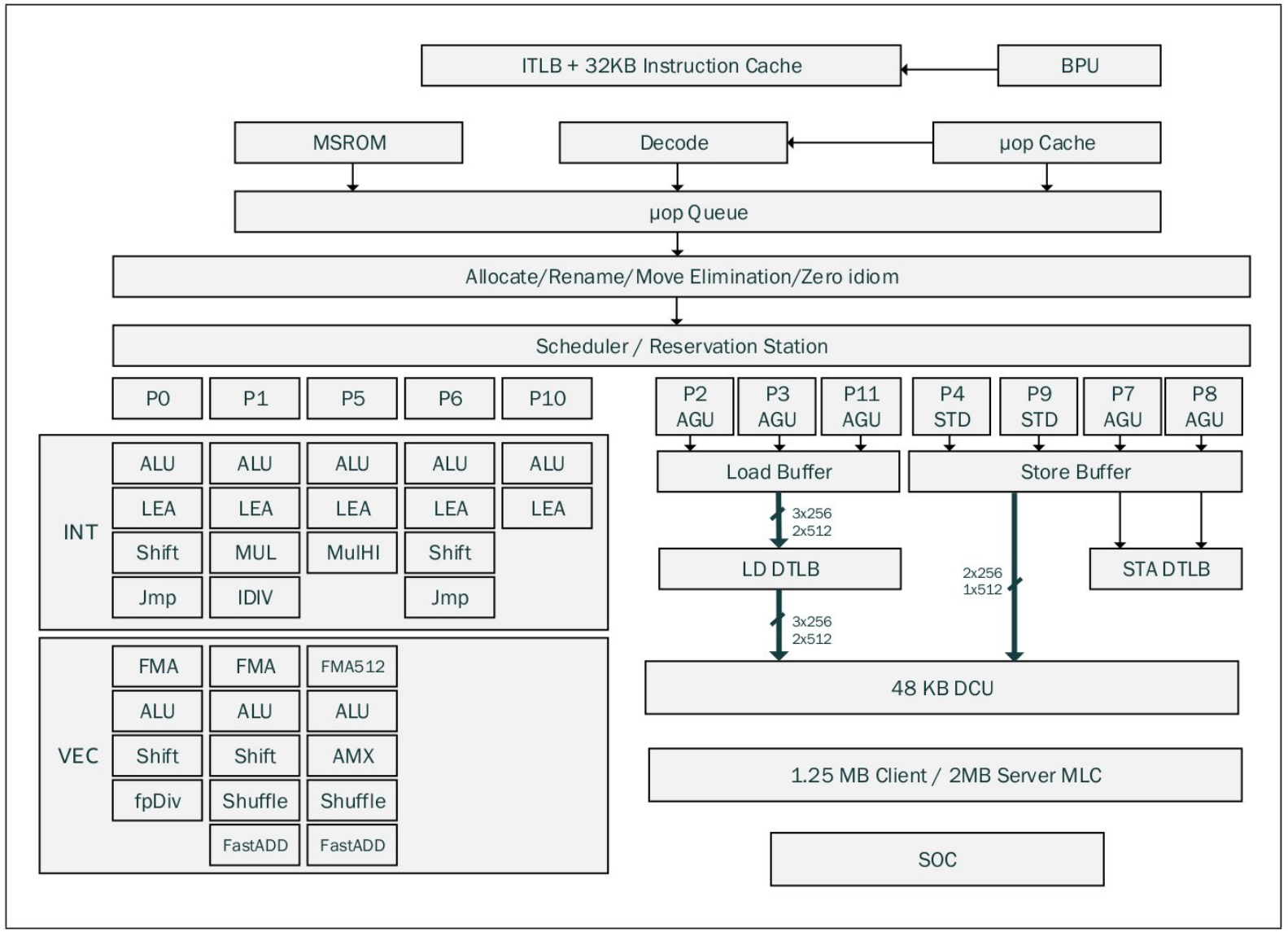
Курс «Архитектурно-ориентированная оптимизация кода»

Intel Top-Down Microarchitecture Analysis Method

Михаил Курносов

Golden Cove Microarchitecture



- Какая из подсистем может ограничивать производительность приложения?

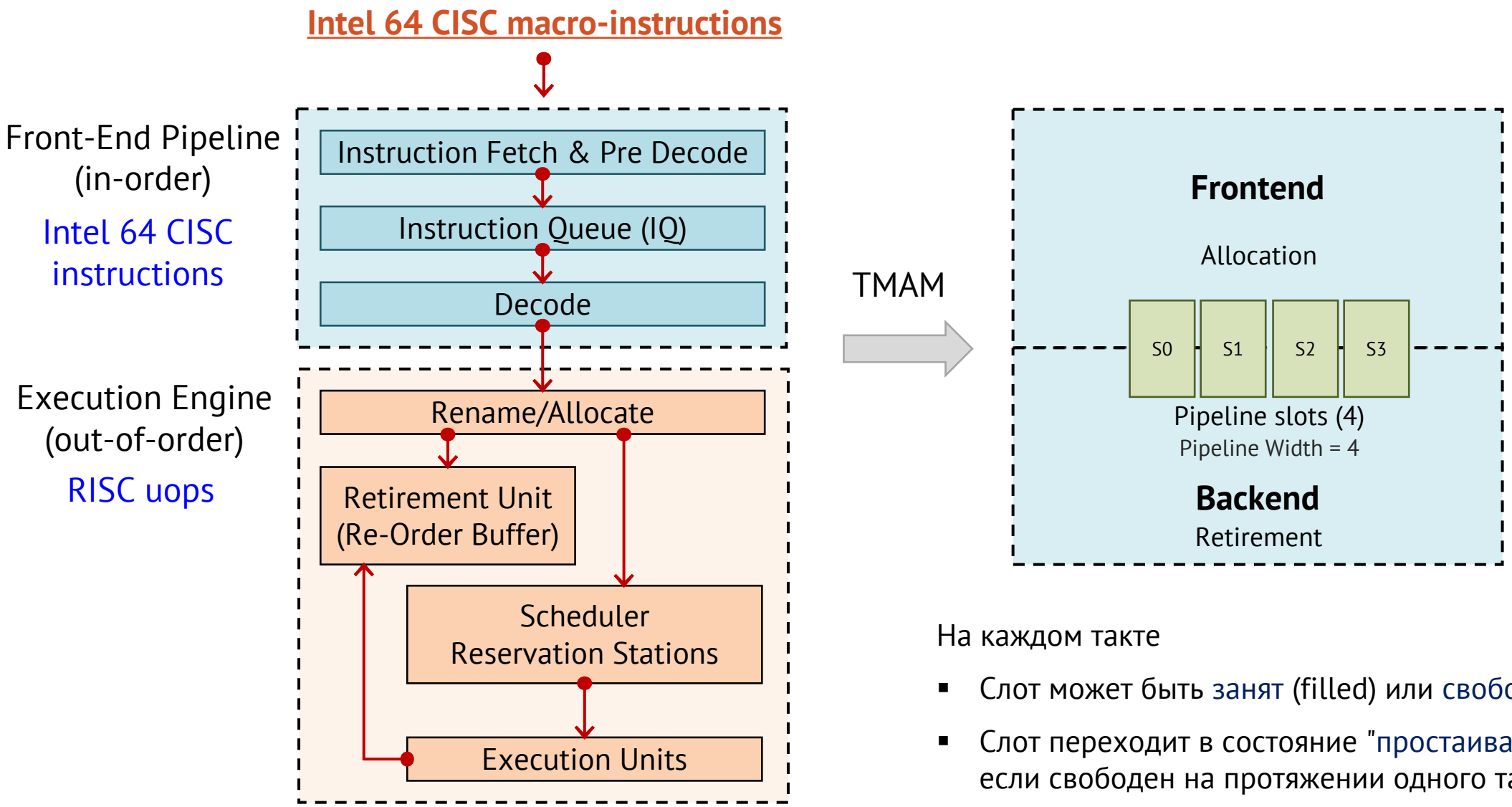
Intel 64 and IA-32 Architectures Optimization Reference Manual

Top-Down Microarchitecture Analysis Method (TMAM)

- **Top-Down Microarchitecture Analysis Method** (TMAM) — метод анализа и оптимизации приложений для микроархитектур Intel по результатам профилирования с ипользованием счетчиков производительности (Performance Monitoring Unit — PMU)
- Метод позволяет определить для заданного приложения узкие места суперскалярного ядра с внеочередным выполнением инструкций
- **Основные понятия:**
  - Pipeline — совокупность всех блоков процессорного узла (Frontend + Backend)
  - Frontend: выбирает и декодирует CISC инструкции Intel 64 в одну или несколько RISC микроопераций (uop)
  - Backend: отслеживает готовность операндов микрооперации и выполняет ее на доступных модулях (портах)
  - Allocation: процесс передачи микрооперации в Backend
  - Retirement: процесс записи результатов микрооперации в архитектурное состояние (в регистры, память)
  - Pipeline slot — совокупность аппаратных ресурсов ядра для выполненя одной uop
- Некоторые операции могут не дойти до фазы Retirement, например отменены из-за неправильного направления спекулятивного выполнения

- Intel 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1
- Ahmad Yasin et al. *A Top-Down method for Performance Analysis and Counters Architecture* // ISPASS, 2014, DOI: 10.1109/ISPASS.2014.6844459

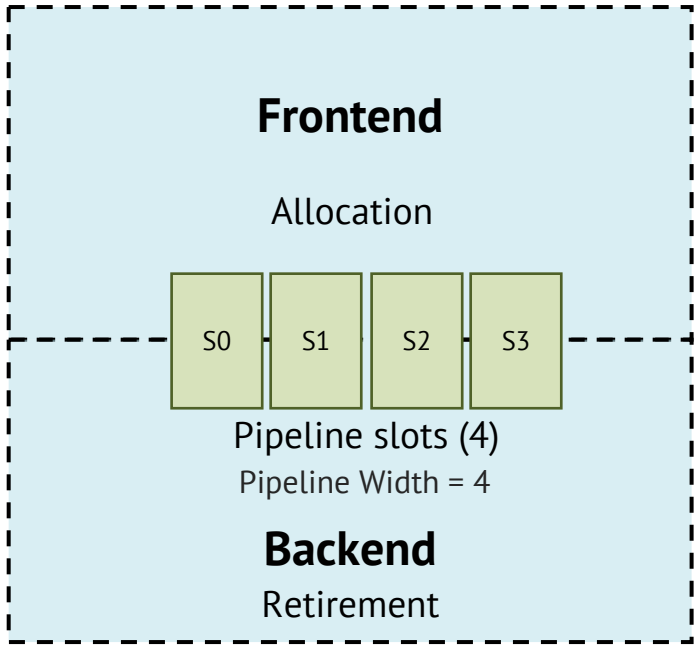
Top-Down Microarchitecture Analysis Method (TMAM)



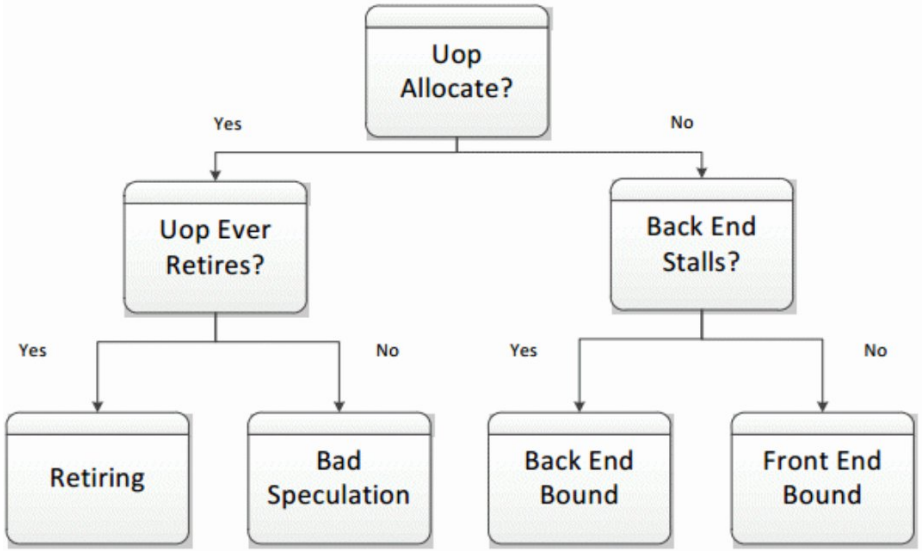
На каждом такте

- Слот может быть занят (filled) или свободен (empty)
- Слот переходит в состояние "простаивает" (stall) если свободен на протяжении одного такта

Top-Down Microarchitecture Analysis Method (TMAM)

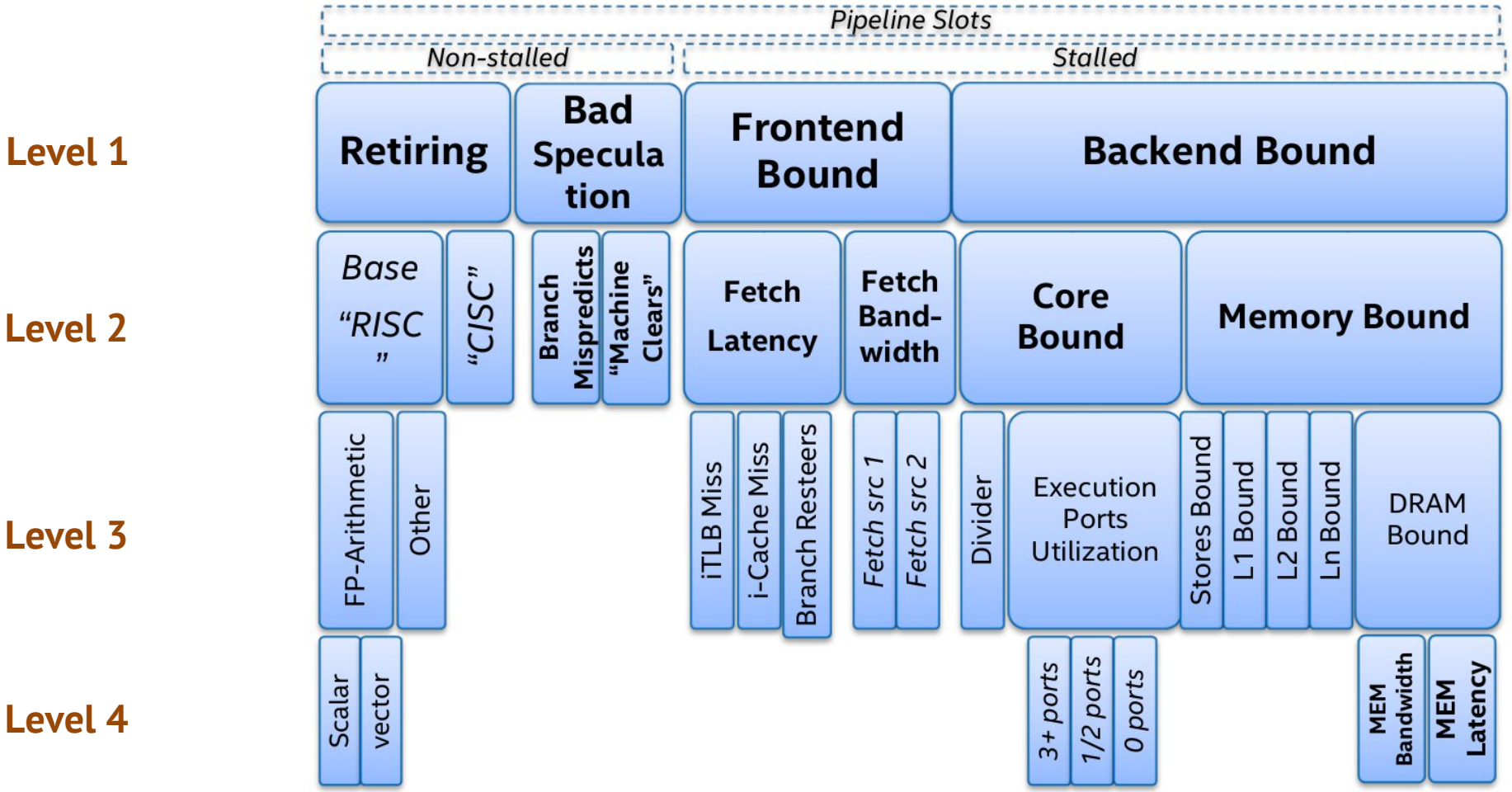


- TMAM позволяет выявить причину простоя слота конвейера
- Слоту назначается класс:
  - Front-End Bound — Frontend не может (не успевает) заполнить слот
  - Back-End Bound — Frontend содержит uop, но не может поместить ее в слот, так как Backend не имеет свободных ресурсов
  - Retiring — слот заполнен и выдает завершенные микрооперации
  - Bad Speculation — слот заполнен, но не выдает завершенные микрооперации (ошибка предсказания перехода, сброс конвейера)



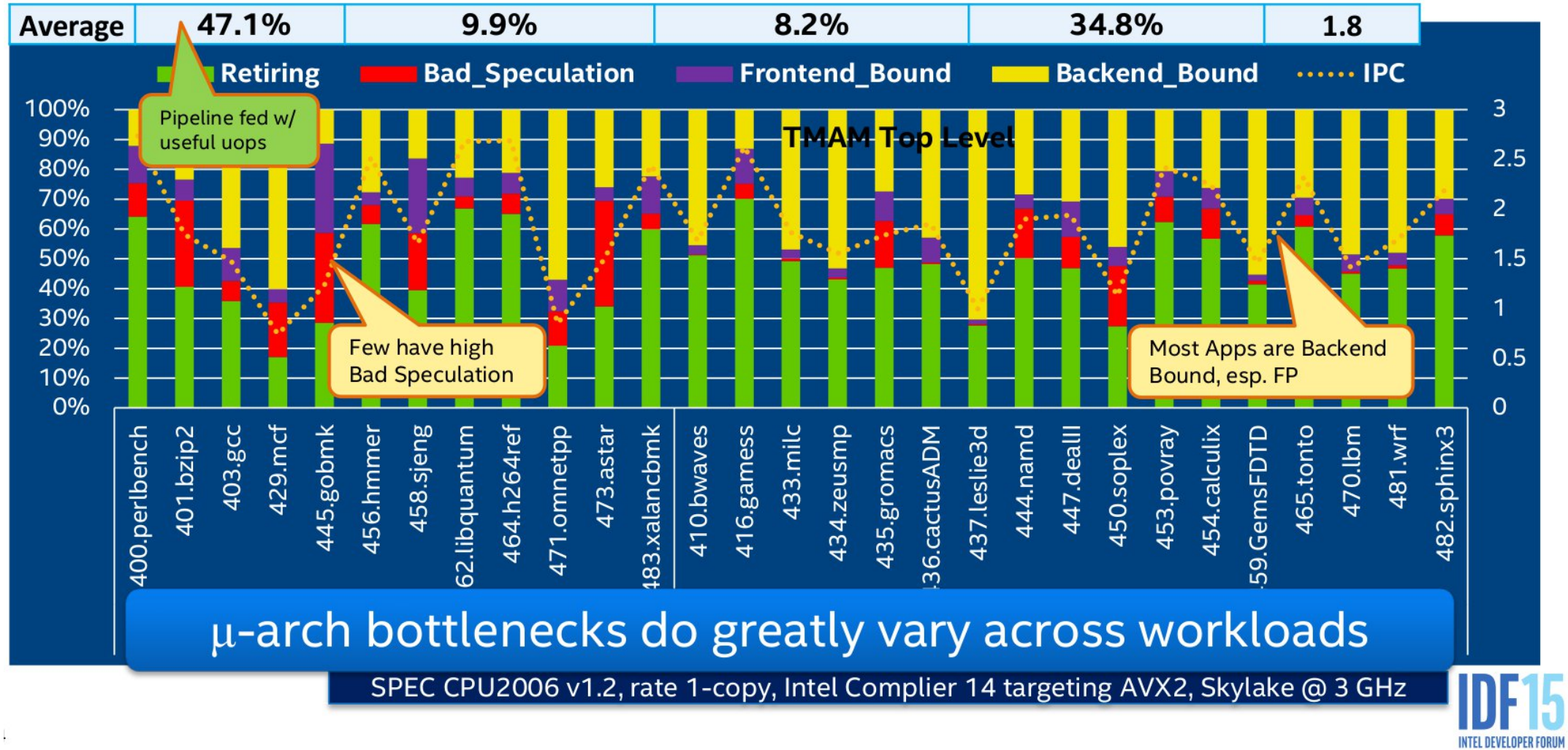
Top-Down Microarchitecture Analysis Method (TMAM)

Четыре уровня классов состояния слота конвейера
(нижние уровни уточняют причины состояния слота)



[*] Ahmad Yasin. How TMA Addresses Challenges in Modern Servers and Enhancements Coming in Ice Lake // Scalable Tools Workshop, 2018

Результаты профилирования приложений различных классов



Ahmad Yasin. Performance Analysis in Modern Multicores // Practical Parallel Programming Course, CS @ Haifa University, 2017

Top-Down Microarchitecture Analysis Method (TMAM)

Результаты профилирования приложений различных классов
(ориентиры для целей оптимизации)

Category	Expected Range of Pipeline Slots in This Category, for a Hotspot in a Well-Tuned:		
	Client/Desktop Application	Server/Database/Distributed application	High Performance Computing (HPC) application
Retiring	20-50%	10-30%	30-70%
Back-End Bound	20-40%	20-60%	20-40%
Front-End Bound	5-10%	10-25%	5-10%
Bad Speculation	5-10%	5-10%	1-5%

https://www.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/top-down-microarchitecture-analysis-method.html

Пример: суммирование элементов массива

```
enum {N = 100000000};

int vec_sum(int *vec, int n)
{
    int s = 0;
    for (int i = 0; i < n; i++) {
        s += vec[i];
    }
    return s;
}

int main()
{
    int *vec = malloc(sizeof(*vec) * N);
    for (int i = 0; i < N; i++) {
        vec[i] = 1;
    }

    double t = wtime();
    int s = vec_sum(vec, N);
    t = wtime() - t;

    printf("Time %.6f, sum %d\n", t, s);
    free(vec);
    return 0;
}
```

Пример: суммирование элементов массива (без оптимизации, level 1)

```
$ gcc -g -fopt-info -o prog ./prog.c

$ toplev --user --core S0-C0 -l1 -v --no-desc taskset -c 0 ./prog
Time 0.201130, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE              Frontend_Bound   % Slots                        2.0  <
BAD             Bad_Speculation  % Slots                        1.2  <
BE              Backend_Bound    % Slots                       71.3   <==
RET             Retiring         % Slots                       25.5  <
MUX                              %                           100.00
Run toplev --describe Backend_Bound^ to get more information on bottleneck
Add --run-sample to find locations
Add --nodes '!+Backend_Bound*/2,+MUX' for breakdown.
```

- -l1 # сбор и анализ счетчиков TMAM уровня 1
- --user # учет событий пространства пользователя
- --core # вывод результатов только для указанных сокет-ядро-аппаратный поток (Sx-Cx-Tx)

- 71% слотов за время выполнения программы были классифицированы как Backend Bound

Пример: суммирование элементов массива (без оптимизации, level 2)

```
$ toplev --user --core S0-C0 -l2 -v --no-desc taskset -c 0 ./prog
Time 0.201266, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE              Frontend_Bound                          % Slots                         1.2  < [25.2%]
BAD             Bad_Speculation                         % Slots                         8.1  < [25.2%]
BE              Backend_Bound                           % Slots                        67.7    [25.2%]<==
RET             Retiring                                % Slots                        23.1  < [74.8%]
FE              Frontend_Bound.Fetch_Latency            % Slots                         0.6  < [25.2%]
FE              Frontend_Bound.Fetch_Bandwidth          % Slots                         0.6  < [25.2%]
BAD             Bad_Speculation.Branch_Mispredicts      % Slots                         0.1  < [25.2%]
BAD             Bad_Speculation.Machine_Clears          % Slots                         8.0  < [25.2%]
BE/Mem          Backend_Bound.Memory_Bound              % Slots                        34.1    [25.2%]
BE/Core         Backend_Bound.Core_Bound                % Slots                        33.6    [25.2%]
RET             Retiring.Light_Operations               % Slots                        18.0  < [25.2%]
RET             Retiring.Heavy_Operations               % Slots                         5.1  < [25.2%]
MUX                                                     %                             25.18
warning: 2 results not referenced: 9 10
Run toplev --describe Backend_Bound^ to get more information on bottleneck
Add --run-sample to find locations
Add --nodes '!+Backend_Bound*/2,+MUX' for breakdown.
```

- 34% слотов за время выполнения классифицированы как Backend Bound/Memory_Bound
- 34% слотов за время выполнения классифицированы как Backend Bound/Core_Bound

Пример: суммирование элементов массива (без оптимизации, level 2)

```
$ toplev --user --core S0-C0 -l2 -v --no-desc taskset -c 0 ./prog
Time 0.201266, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE              Frontend_Bound                          % Slots                         1.2  < [25.2%]
BAD             Bad_Speculation                         % Slots                         8.1  < [25.2%]
BE              Backend_Bound                           % Slots                        67.7    [25.2%]<==
RET             Retiring                                % Slots                        23.1  < [74.8%]
FE              Frontend_Bound.Fetch_Latency            % Slots                         0.6  < [25.2%]
FE              Frontend_Bound.Fetch_Bandwidth          % Slots                         0.6  < [25.2%]
BAD             Bad_Speculation.Branch_Mispredicts      % Slots                         0.1  < [25.2%]
BAD             Bad_Speculation.Machine_Clears          % Slots                         8.0  < [25.2%]
BE/Mem          Backend_Bound.Memory_Bound              % Slots                        34.1    [25.2%]
BE/Core         Backend_Bound.Core_Bound                % Slots                        33.6    [25.2%]
RET             Retiring.Light_Operations               % Slots                        18.0  < [25.2%]
RET             Retiring.Heavy_Operations               % Slots                         5.1  < [25.2%]
MUX                                                     %                             25.18
warning: 2 results not referenced: 9 10
Run toplev --describe Backend_Bound^ to get more information on bottleneck
Add --run-sample to find locations
Add --nodes '!+Backend_Bound*/2,+MUX' for breakdown.
```

- 34% слотов за время выполнения классифицированы как Backend Bound/Memory_Bound
- 34% слотов за время выполнения классифицированы как Backend Bound/Core_Bound

Пример: суммирование элементов массива (без оптимизации, level 3)

```
$ toplev --user --core S0-C0 -l3 -v --no-desc taskset -c 0 ./prog
Time 0.200856, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE             Frontend_Bound                                % Slots                  1.2  < [ 6.4%]
BAD            Bad_Speculation                               % Slots                  5.4  < [ 6.4%]
BE             Backend_Bound                                 % Slots                 69.4    [ 5.8%]<==
RET            Retiring                                      % Slots                 24.0  < [11.8%]
BE/Mem         Backend_Bound.Memory_Bound                    % Slots                 34.0    [ 5.8%]
BE/Core        Backend_Bound.Core_Bound                      % Slots                 35.4    [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L1_Bound           % Stalls                24.0    [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L2_Bound           % Stalls                 0.4  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L3_Bound           % Stalls                 0.0  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.DRAM_Bound         % Stalls                 0.2  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.Store_Bound        % Stalls                 0.0  < [ 5.8%]
BE/Core        Backend_Bound.Core_Bound.Divider              % Clocks                 0.0  < [ 5.8%]
BE/Core        Backend_Bound.Core_Bound.Ports_Utilization    % Clocks                36.9    [ 5.8%]
RET            Retiring.Light_Operations.FP_Arith            % Uops                   0.0  < [ 5.8%]
RET            Retiring.Light_Operations.Memory_Operations   % Slots                 11.2  < [ 5.8%]
RET            Retiring.Light_Operations.Branch_Instructions % Slots                  2.0  < [ 5.8%]
RET            Retiring.Light_Operations.Nop_Instructions    % Slots                  0.0  < [ 5.8%]
RET            Retiring.Light_Operations.Other_Light_Ops     % Slots                  5.4  < [ 5.8%]
RET            Retiring.Heavy_Operations.Few_Uops_Instructions % Slots                4.8  < [ 5.8%]
RET            Retiring.Heavy_Operations.Microcode_Sequencer % Slots                  0.6  < [ 5.8%]
MUX                                                          %                        5.84
```

Пример: суммирование элементов массива (без оптимизации, level 3)

```
$ toplev --user --core S0-C0 -l3 -v --no-desc taskset -c 0 ./prog
Time 0.200856, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE             Frontend_Bound                                % Slots                  1.2  < [ 6.4%]
BAD            Bad_Speculation                               % Slots                  5.4  < [ 6.4%]
BE             Backend_Bound                                 % Slots                 69.4    [ 5.8%]<==
RET            Retiring                                      % Slots                 24.0  < [11.8%]
BE/Mem         Backend_Bound.Memory_Bound                    % Slots                 34.0    [ 5.8%]
BE/Core        Backend_Bound.Core_Bound                      % Slots                 35.4    [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L1_Bound           % Stalls                24.0    [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L2_Bound           % Stalls                 0.4  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.L3_Bound           % Stalls                 0.0  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.DRAM_Bound         % Stalls                 0.2  < [ 5.8%]
BE/Mem         Backend_Bound.Memory_Bound.Store_Bound        % Stalls                 0.0  < [ 5.8%]
BE/Core        Backend_Bound.Core_Bound.Divider              % Clocks                 0.0  < [ 5.8%]
BE/Core        Backend_Bound.Core_Bound.Ports_Utilization    % Clocks                36.9    [ 5.8%]
RET            Retiring.Light_Operations.FP_Arith            % Uops                   0.0  < [ 5.8%]
RET            Retiring.Light_Operations.Memory_Operations   % Slots                 11.2  < [ 5.8%]
RET            Retiring.Light_Operations.Branch_Instructions % Slots                  2.0  < [ 5.8%]
RET            Retiring.Light_Operations.Nop_Instructions    % Slots                  0.0  < [ 5.8%]
RET            Retiring.Light_Operations.Other_Light_Ops     % Slots                  5.4  < [ 5.8%]
RET            Retiring.Heavy_Operations.Few_Uops_Instructions % Slots                4.8  < [ 5.8%]
RET            Retiring.Heavy_Operations.Microcode_Sequencer % Slots                  0.6  < [ 5.8%]
MUX                                                          %                        5.84
```

Пример: суммирование элементов массива (-O2, level 1)

```
$ gcc -g -mavx512f -mtune=native -O2 -fopt-info -o prog ./prog.c
./prog.c:32:13: optimized:  Inlining vec_sum/40 into main/41.
./prog.c:18:23: optimized: loop vectorized using 32 byte vectors
./prog.c:27:23: optimized: loop vectorized using 32 byte vectors
```

```
$ toplev --user --core S0-C0 -l1 -v --no-desc taskset -c 0 ./prog
Time 0.019192, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE              Frontend_Bound   % Slots                       13.3  <
BAD             Bad_Speculation  % Slots                        5.2  <
BE              Backend_Bound    % Slots                       62.4   <==
RET             Retiring         % Slots                       19.1  <
MUX                              %                           100.00
```

Пример: суммирование элементов массива (-O2, level 2)

```
$ toplev --user --core S0-C0 -l2 -v --no-desc taskset -c 0 ./prog
Time 0.019184, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE             Frontend_Bound                          % Slots                        9.1  < [26.8%]
BAD            Bad_Speculation                         % Slots                       10.4  < [26.8%]
BE             Backend_Bound                           % Slots                       66.3    [26.1%]<==
RET            Retiring                                % Slots                       14.2  < [73.2%]
FE             Frontend_Bound.Fetch_Latency            % Slots                        6.3  < [26.8%]
FE             Frontend_Bound.Fetch_Bandwidth          % Slots                        2.9  < [26.8%]
BAD            Bad_Speculation.Branch_Mispredicts      % Slots                        0.5  < [26.8%]
BAD            Bad_Speculation.Machine_Clears          % Slots                        9.9  < [26.8%]
BE/Mem         Backend_Bound.Memory_Bound              % Slots                       33.5    [26.1%]
BE/Core        Backend_Bound.Core_Bound                % Slots                       32.8    [26.1%]
RET            Retiring.Light_Operations               % Slots                        9.1  < [26.8%]
RET            Retiring.Heavy_Operations               % Slots                        5.1  < [26.8%]
MUX                                                    %                            26.14
```

Пример: суммирование элементов массива (-O3, level 1)

```
$ gcc -g -mavx512f -mtune=native -O3 -fopt-info -o prog ./prog.c
./prog.c:138:13: optimized:  Inlining vec_sum/40 into main/44.
./prog.c:18:23: optimized: loop vectorized using 32 byte vectors
./prog.c:18:23: optimized: loop with 6 iterations completely unrolled (header execution count 46365075)

$ toplev --user --core S0-C0 -l1 -v --no-desc taskset -c 0 ./prog
Time 0.019268, sum 100000000
# 4.4-full-perf on 11th Gen Intel(R) Core(TM) i7-1160G7 @ 1.20GHz [tgl/icelake]
FE             Frontend_Bound   % Slots                       10.2  <
BAD            Bad_Speculation  % Slots                        5.9  <
BE             Backend_Bound    % Slots                       68.2   <==
RET            Retiring         % Slots                       15.7  <
MUX                             %                           100.00
```

Пример: суммирование элементов массива (-O3, level 1)

```
$ perf stat --topdown -- taskset -c 0 ./prog
Time 0.022320, sum 100000000

 Performance counter stats for 'taskset -c 0 ./prog':

            retiring      bad speculation       frontend bound        backend bound
               29,0%                 4,3%                17,3%                49,4%

       0,124534287 seconds time elapsed

       0,056246000 seconds user
       0,068299000 seconds sys
```

Пример: умножение матриц (DGEMM)

```
# 11th Gen Intel Core i7 1160G7 [tgl/icelake],  GNU/Linux 5.19.0-23-generic
$ gcc -mtune=native -fopt-info -g -Wall -o dgemm ./dgemm.c
```

```c
#define N 1024
__attribute__((target("sse"), optimize("O0")))
void dgemm_def(double *a, double *b, double *c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                c[i * n + j] += a[i * n + k] * b[k * n + j];
            }
        }
    }
}
```

```
Backend_Bound.Memory_Bound  % Slots                 35.9    <==
Backend_Bound.Core_Bound    % Slots                 23.9
```

Версия	Время (сек)	GFLOPS	IPC	CPI	Ускорение	Frontend Bound	Bad Speculation	Backend Bound	Retiring
Default (-O0, sse)	4.9458	0.43	2.08	0.48	1.0	1.6	1.2	**59.6**	37.6

Пример: умножение матриц (DGEMM)

```
__attribute__((target("sse"), optimize("O0")))
void dgemm_loop_xchg(double *a, double *b, double *c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            for (int j = 0; j < n; j++) {
                c[i * n + j] += a[i * n + k] * b[k * n + j];
            }
        }
    }
}
```

```
Backend_Bound.Memory_Bound  % Slots                    13.7  <
Backend_Bound.Core_Bound    % Slots                    23.5   <==
```

Версия	Время (сек)	GFLOPS	IPC	CPI	Ускорение	Frontend Bound	Bad Speculation	Backend Bound	Retiring
Default (-O0, sse)	4.9458	0.43	2.08	0.48	1.0	1.6	1.2	59.6	37.6
Loop Interchange (-O0, sse)	2.7932	0.77	2.57	0.39	1.8	2.6	0.8	**48.5**	48.1

Пример: умножение матриц (DGEMM)

```
__attribute__((target("sse"), optimize("O2", "-ftree-vectorize")))
void dgemm_vec(double * a, double *b, double *c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            #pragma GCC ivdep
            for (int j = 0; j < n; j++) {
                c[i * n + j] += a[i * n + k] * b[k * n + j];
            }
        }
    }
}
```

```
Backend_Bound.Memory_Bound  % Slots                 20.5
Backend_Bound.Core_Bound    % Slots                 30.8    <==
```

Версия	Время (сек)	GFLOPS	IPC	CPI	Ускорение	Frontend Bound	Bad Speculation	Backend Bound	Retiring
Default (-O0, sse)	4.9458	0.43	2.08	0.48	1.0	1.6	1.2	59.6	37.6
Loop Interchange (-O0, sse)	2.7932	0.77	2.57	0.39	1.8	2.6	0.8	48.5	48.1
Vectorized SSE (-O2, sse, −ftree-vectorize)	0.3451	6.22	1.95	0.51	14.3	2.0	1.2	**58.8**	38.0

Пример: умножение матриц (DGEMM)

```
__attribute__((target("avx2"), optimize("O2", "-ftree-vectorize")))
void dgemm_vec_avx2(double * a, double *b, double *c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            #pragma GCC ivdep
            for (int j = 0; j < n; j++) {
                c[i * n + j] += a[i * n + k] * b[k * n + j];
            }
        }
    }
}
```

```
Backend_Bound.Memory_Bound  % Slots                    18.5  <
Backend_Bound.Core_Bound    % Slots                    33.7   <==
```

Версия	Время (сек)	GFLOPS	IPC	CPI	Ускорение	Frontend Bound	Bad Speculation	Backend Bound	Retiring
Default (-O0, sse)	4.9458	0.43	2.08	0.48	1.0	1.6	1.2	59.6	37.6
Loop Interchange (-O0, sse)	2.7932	0.77	2.57	0.39	1.8	2.6	0.8	48.5	48.1
Vectorized SSE (-O2, sse, –ftree-vectorize)	0.3451	6.22	1.95	0.51	14.3	2.0	1.2	58.8	38.0
Vectorized AVX2 (-O2, avx2, –ftree-vectorize)	0.2111	10.17	1.98	0.51	23.4	2.0	0.8	**62.0**	35.3

Ссылки

- Intel 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1 (TMAM)
- Ahmad Yasin et al. **A Top-Down method for Performance Analysis and Counters Architecture** // ISPASS, 2014, DOI: 10.1109/ISPASS.2014.6844459
- **Toplev (PMU-Tools)** // https://github.com/andikleen/pmu-tools
- **TMA Performance Metrics** // https://download.01.org/perfmon/TMA_Metrics.xlsx