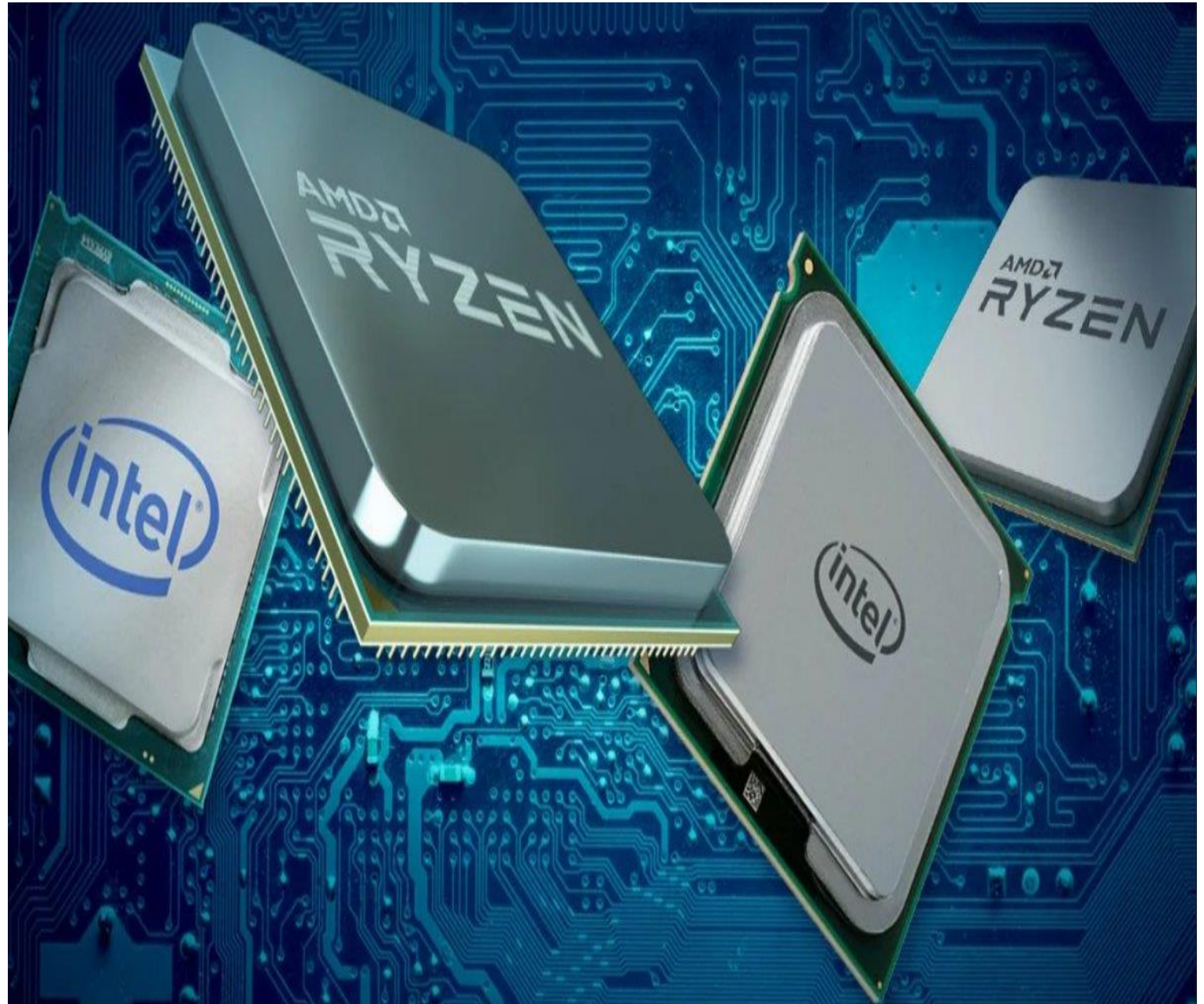


Повышение производительности прикладного ПО общего назначения процессоров x86_64 серии путем доработки системного ПО

УНИВЕРСАЛЬНЫЙ СПОСОБ
УСКОРЕНИЯ ВЫЧИСЛЕНИЙ
НА ЭТАПЕ ИСПОЛНЕНИЯ



Метрики продуктивности

Все знают про коэффициент полезного действия теплового двигателя:

КПД паровоза = 5–12%;

КПД бензинового двигателя внутреннего сгорания (далее – ДВС) = 25–30%;

КПД дизельного ДВС = 35–40%;

Максимальный КПД реального ДВС = 50% (турбонаддув, охлаждение рабочего тела);

КПД газотурбинной установки электрогенератора большой мощности = 25–38%.

КПД реактивного двигателя = 25%.

Исходная продуктивность программного кода, разработанного с помощью современных систем программирования составляет 2,5–10%! Это хуже КПД паровоза и это никого не смущает в эпоху бурного развития технологий с использованием искусственного интеллекта (далее – ИИ). Далеко ли мы уедем на вычислительном «паровозе» в эпоху ИИ?

Эволюция исходной продуктивности программного кода, накладных расходов по поколениям ЭВМ, технологий программирования

Поколения ЭВМ	Исходная продуктивность программного кода %	Накладные расходы %	Технологии программирования
II поколение (транзисторы)	33	67	Ручная, в машинных кодах
III поколение (ИС и БИС)	20	80	PL1, Fortran, функциональное программирование, + стековая память
IV поколение (СБИС, процессоры), пользовательский режим	10	90	ООП(объектно-ориентированное программирование - дескрипторы) , стековая память
IV поколение (СБИС, процессоры), режим ядра	2,5	97,5	Ассемблер, ООП, стековая память

История вопроса

- В 90-е годы прошлого века автор занимался оценкой производительности вычислительных комплексов (далее – ВК) и являлся разработчиком типовых методик оценки производительности ВК АСУ ПВО бригадного(корпусного) звена и выше.
- . ВК создавали большие коллективы инженеров программистов и когда программно-аппаратный комплекс приходил на испытания, тогда выяснялось, что производительности ВК недостаточно для решения объема задач, заданных в ТТЗ. А деньги на проект уже потрачены, переписывать программы и дорабатывать ВК уже некогда, нужно принимать на вооружение, а он не соответствует ТТЗ. Необходимо было провести в кратчайшие сроки минимальные доработки, чтобы ВК АСУ ПВО по производительности стал соответствовать ТТЗ (даже библ.им.Ленина Москвы не помогла автору – он не нашел в ней ответа, его еще тогда не существовало!).
- Тогда у автора и возникла идея проверить свою гипотезу ускорить вычисления, сократив накладные расходы, путем изменения организации вычислений (ничего не меняя в остальном комплексе программ). И она сработала! Всего по такой технологии было доработано 4 промышленных ВК АСУ ПВО с эффектом повышения производительности от 2-х до 4-х раз. Акт о реализации есть.

Предпосылки ускорения ВК

- Функциональное ПО не менялось.
- Целевую функцию задач ВК при однократном подключении исполняла меньшая часть программного кода – 33%, а остальные 67% составляли накладные расходы (подготовка запуска целевой функции), т.е. исходная продуктивность кода при обработке одного блока данных составляла 33%. Целевую функцию после подготовки запуска вы можете выполнить как однократно, так и многократно (обработать несколько блоков информации сразу). При блочной обработке продуктивность кода растет, а накладные расходы уменьшаются, что сказывается на сокращении количества команд при обработке нескольких блоков разными способами. Такая оценка (33%) была характерна для программного обеспечения (далее – ПО), разработанного вручную в машинных кодах, для электронно-вычислительной техники (далее ЭВТ) 2-го поколения.
- Иерархия описания функционала программ в системах программирования (объектно – ориентированное программирование) требует при исполнении кода программы определить место расположения кода и данных, сначала прочитать описание программных сущностей и структуры обрабатываемых данных и потом приступить непосредственно к обработке, а затраты на эти действия могут превышать сам арифметический цикл обработки.
- Сама ОС(ядро) это накладные расходы в чистом виде – она получает данные извне на скорости памяти и передает их туда же при необходимости, передает данные приложению и забирает результаты вычислений, но уже с использованием иерархии кэш-памяти. Обмен этот формализован, но в зависимости от быстродействия элементов инфраструктуры ПК (диск, память, кэш, процессор, сеть) одно и тоже взаимодействие будет характеризоваться различным количеством элементарных действий, доходящих по разнице по количеству до порядков (даже без учета недеklarированных и недокументированных возможностей ПО). Контроля за этим процессом с точки зрения затрат ресурсов не существует.
- Большая вложенность вызовов 16–18 ЭВТ 4 поколения и необходимость сохранять и восстанавливать регистры процессора при вызове и возврате, а целевая функция исполняется на максимальной вложенности и повторы исполняются на максимальных уровнях вложенности, не возвращаясь обратно на исходный уровень. Дальний вызов (за пределы 64К и возврат(команды Call->Ret ассемблера)) требует 17 команд PUSH на сохранение регистров процессора x86_64 и 17 команд POP на их восстановление при возврате управления(всего 34).
- В процессорах X86-64 серии есть команда syscall, которая выполняет переход в режим ядра в пользовательском пространстве для выполнения функций ядра, с упомянутым выше сохранением регистров, но при этом полностью зачищает все данные от пользовательского режима (конвейер, наработанную таблицу переходов, регистры сегментов и т.д и т.п.) и при возврате в пользовательский режим «необученному» процессору приходится медленно запускать конвейер и еще долго восстанавливать все атрибуты.
- Эволюция продуктивности ПО, разработанного с помощью систем программирования с ООП для современных ЭВТ 4 поколения, составляет 2,5–10% и никого этот факт не смущает! Предлагаемый способ повышения производительности ПО позволяет увеличить реальное быстродействие процессоров на 1–2 порядка путем восстановления продуктивности ПО на этапе исполнения как минимум до 50% – это точка перегиба. Продуктивность можно восстанавливать и от 50 до 100%, но уже с меньшей эффективностью.
- До недавнего времени русскоязычные публикации по проблеме накладных расходов принадлежали только мне. В октябре 2023 года в журнале «Молодой ученый» (№43(490)/2023) появилась первая публикация по снижению накладных расходов белорусских товарищей – Селезнёв А.И., Селезнёв И.Л. «Контейнеризация в системах обработки данных». Технология контейнеризации при создании контейнеров уменьшают накладные расходы(вместо использования виртуальных машин – количественные оценки уменьшения накладных расходов не приводятся!?), а применение моей технологии на этапе исполнения контейнеров, позволит еще как минимум на порядок поднять ее эффективность! Интерес к решениям на основе контейнеров и их широкое использование привело к тому, что в 2015 году компания Docker и другие лидеры отрасли в рамках проекта Open Container Initiative (OCI) разработали стандарты упаковки программного кода в контейнеры.

№ _____
Гос. _____
1—1

А К Т

научно-технической комиссии о реализации научных положений и выводов кандидатской диссертации СУВОРОВА Дмитрия Анатольевича

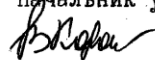
Комиссия в составе: председатель-начальник управления кандидат военных наук старший научный сотрудник В.М.КОРОЛЕВ; члены комиссии-начальник отдела П.А.ДЕРИЙ, начальник лаборатории кандидат технических наук старший научный сотрудник В.В.ГОРЯЧКИН, старший научный сотрудник кандидат технических наук А.А.РАЕВ констатирует, что диссертационная работа тов. Д.А.СУВОРОВА на тему: "Расчётно-экспериментальный метод оценки и повышения пропускной способности вычислительных комплексов АСУ реального времени" по своему содержанию соответствует направленности работ

Исследования по теме диссертационной работы были направлены на развитие и повышение достоверности моделей оценки производительности вычислительных комплексов, повышение эффективности использования вычислительных ресурсов в АСУ реального времени. В ходе исследований выявлены новые зависимости характеристик обслуживания заявок от временных факторов и разработаны их модели, введены понятия и расчётные соотношения для накладных расходов и непроизводительных затрат процессорного времени. Их использование в разработанном расчётно-экспериментальном методе позволило повысить точность оценки показателей производительности и оценить величину непроизводительных затрат процессорного времени при перегрузке вычислительных комплексов по времени счета.

Кроме того, результаты работы использованы при выработке рекомендаций по оптимизации систем организации вычислительного процесса в четырех различных вычислительных комплексах АСУ реального времени,

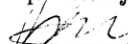
а так же реализованы в НИР "Хамса" и "Буровая".

Реализация разработанных технических решений и рекомендаций по доработкам общего программного обеспечения позволила исключить непроизводительные затраты процессорного времени в размере от двадцати до сорока процентов при полутора-двукратной перегрузке вычислительных комплексов, повысить их пропускную способность в полтора-два раза, обеспечить устойчивость и правильность функционирования специального программного обеспечения при указанных уровнях перегрузки в реальном времени.

Председатель комиссии
Начальник управления
 В.М.КОРОЛЕВ

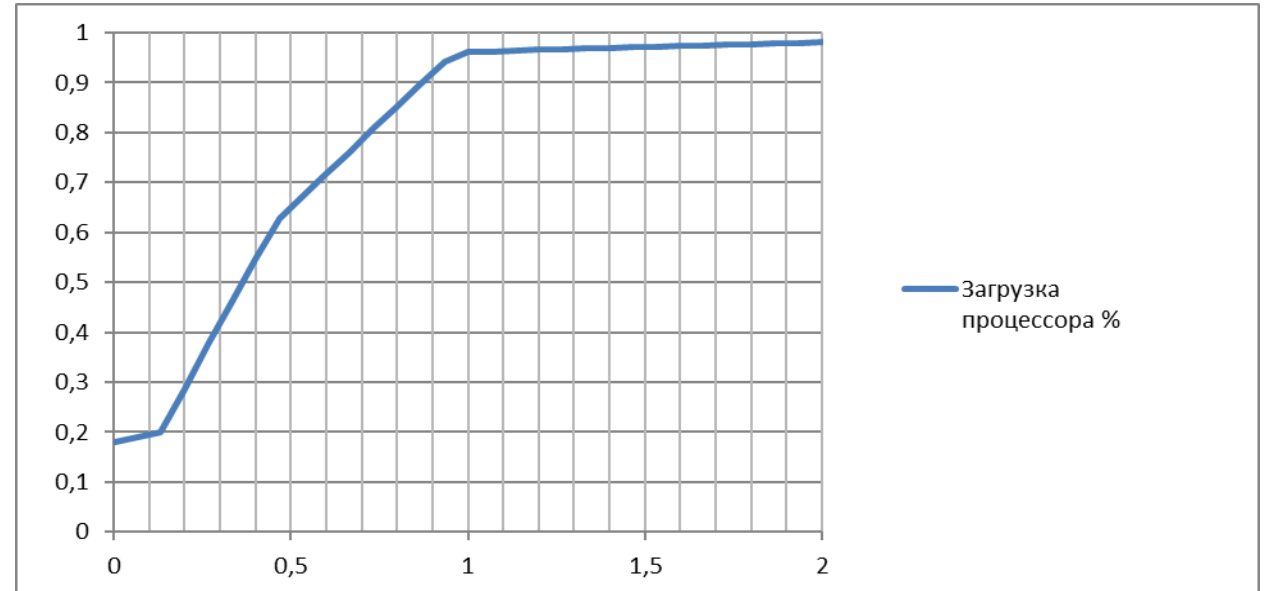
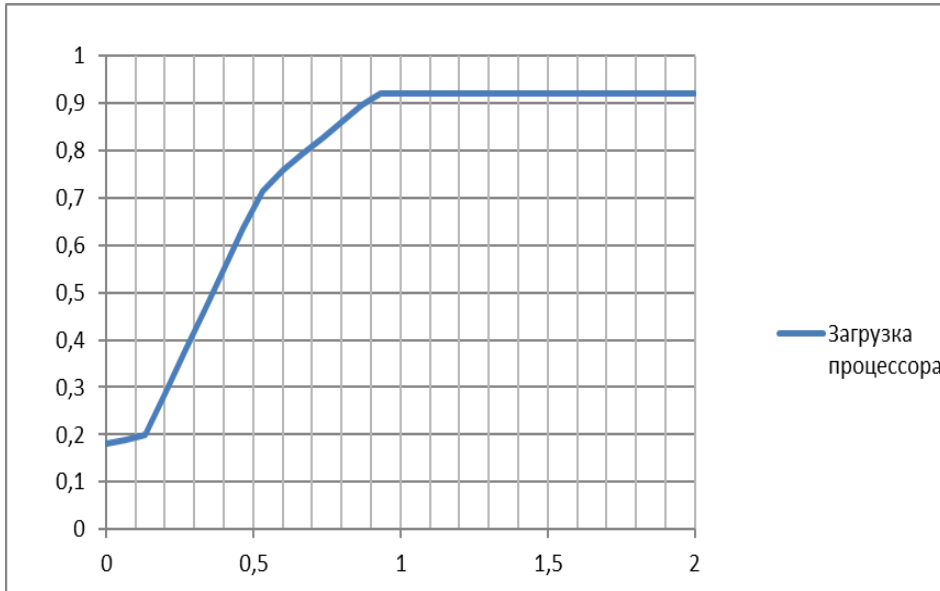
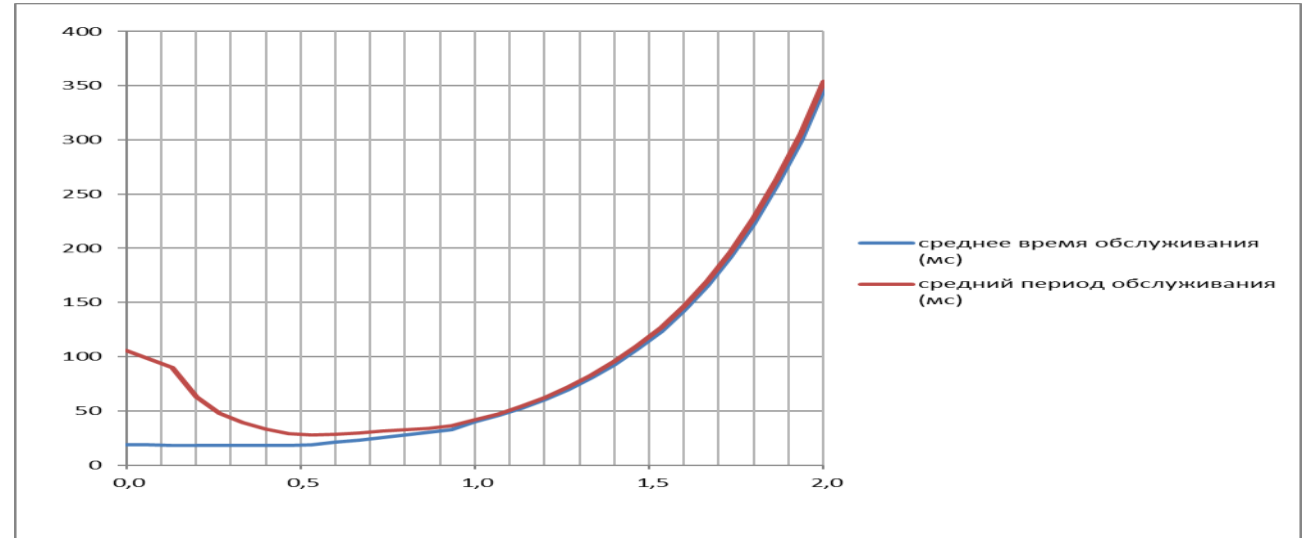
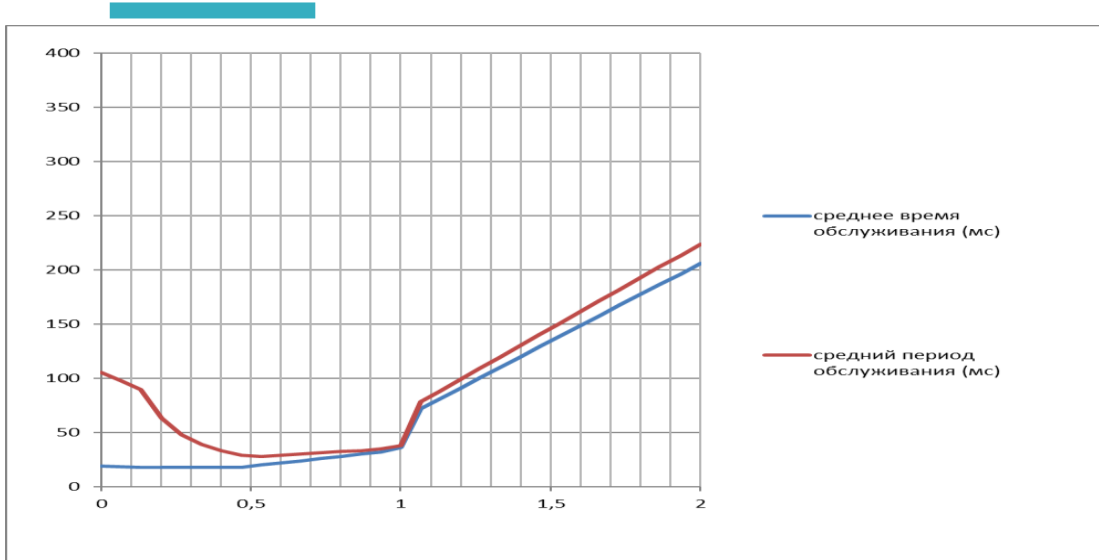
Члены комиссии:
Начальник отдела

 П.А.ДЕРИЙ
Начальник лаборатории
 В.В.ГОРЯЧКИН

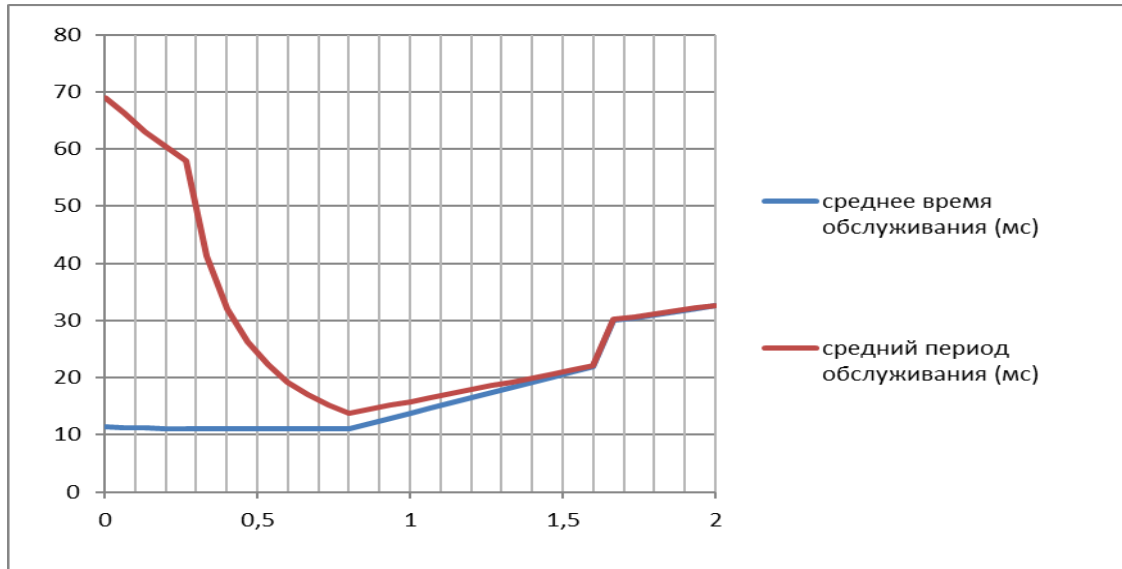
Старший научный сотрудник
 А.А.РАЕВ

" 6 " апреля 1992г.

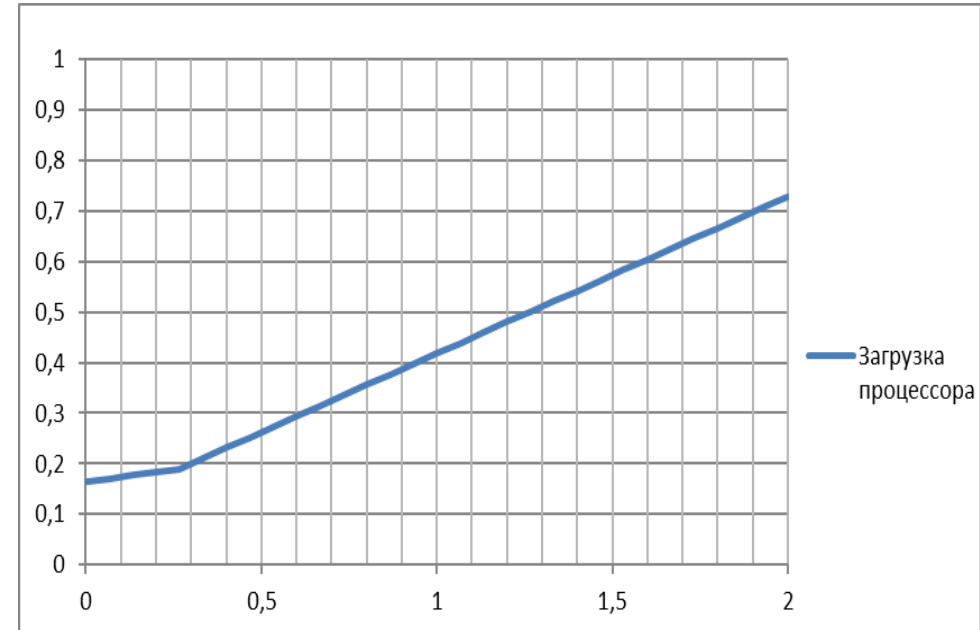
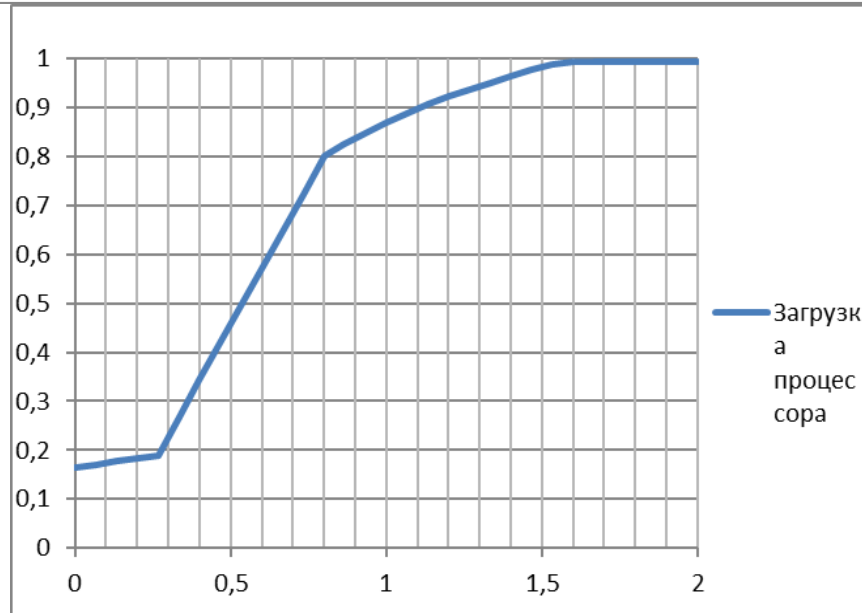
Реализация в промышленной серии прототип II поколения (продуктивность кода увеличена до 96%)



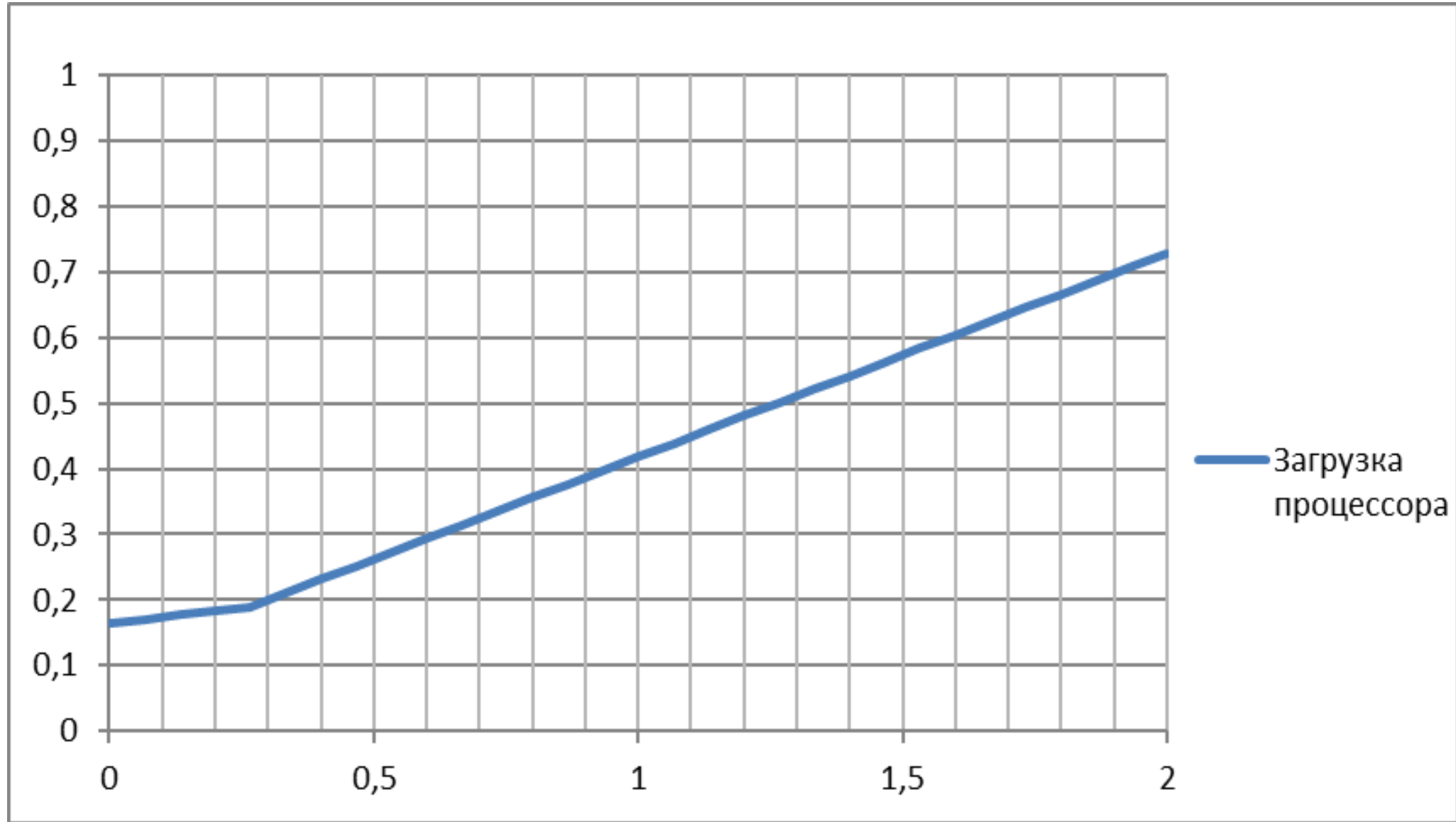
Память в 16 раз быстрее и есть L1 кэш данных ЭВМ III поколения (продуктивность кода увеличена до 50%)



? Ноу-хау автора ?



Для ЭВМ III поколения были проведены все необходимые доработки по способу ускорения вычислений и производительность была повышена в 4 раза и это был не предел (для этого ВК было изменено ТТЗ по пропускной способности в сторону увеличения почти в 2 раза), но этого оказалось достаточно во всем диапазоне загрузок. Задания выжать максимум не стояла, возможность увеличения продуктивности еще больше оставалась (с 50% до 100%). Состояние до доработок приведено на предыдущих слайдах.



Сравнение ЭВМ прототипа и пришедшей на замену

Поколение ЭВМ, тип	II(транзисторы, ДЗУ ферритовое)	III, ЕС ЭВМ ряд 1
Разрядность, бит	29	32
Объем ПО, Мбайт	1	4
Быстродействие, млн. оп.сек	0,225	3(Смесь Гиббсона)
Цикл обращения к памяти, мкс	4,5	0,25
Кэш данных	-	есть

Фрагмент статистики ВК 2 и 3 поколений

Фрагмент статистики ВК 2-го поколения				
№ п.п.	п (кол-во включе- ний)	ρ (загрузка процес- сора) %	τ (время вклю- чения)мс	λ (интен - сивность включе - ния)
1	618	52,9	311	1,68
2	617	10,04	58	1,68
3	1667	11,08	23	4,57
4	583	2,9	18	1,59
5	602	3,48	20	1,64
6	263	1,56	21	0,71
7	427	1,21	10	1,18
8	375	2,12	20	1,04
9	19	1,48	283	0,05
10	618	0,18	1	1,68
11	24	1,71	260	0,06
12	24	0,17	25	0,06
13	19	1,04	198	0,05
14	10	0,59	215	0,03
15	10	0,15	55	0,03
16	10	0,47	169	0,03
17	10	0,31	112	0,03
18	19	1,77	337	0,05
19	10	1,04	419	0,03
	5925	94,2	198,5341	16,19

Фрагмент статистики ВК 3-го поколения				
№ п.п.	п (кол-во включе- ний)	ρ (загрузка процес- сора) %	τ (время вклю- чения)мс	λ (интен - сивность включе - ния)
1	285	9,509	35,072	2,720
2	746	6,104	8,601	7,108
3	192	0,089	0,49	1,824
4	6	0,094	16,471	0,130
5	236	7,113	31,685	2,246
6	472	0,504	1,301	4,489
7	11	0,022	2,104	0,101
8	21	0,02	1,015	0,199
9	4	0,001	0,497	0,034
10	730	6,292	9,061	6,948
11	109	2,153	20,763	1,075
12	21	0,112	5,642	0,198
13	11	0,22	21,082	0,099
14	64	2,116	34,768	0,621
15	32	0,033	1,106	0,307
16	17	0,024	1,525	0,170
17	4	0,092	24,251	0,033
18	21	0,074	3,711	0,200
19	11	0,479	45,803	0,100
20	8	0,245	32,244	0,076
21	2	0,031	16,438	0,019
22	11	0,027	2,673	0,099
23	8	0,256	33,977	0,075
24	11	0,121	11,576	0,098
25	11	0,004	0,439	0,100
26	30	1,141	39,998	0,287
27	11	0,332	31,742	0,100
28	6	0,05	8,896	0,049
29	21	0,22	11,015	0,199
30	51	0,722	14,888	0,484
31	1	0,168	177,411	0,008
32	1	0,025	27,157	0,008
33	13	0,175	14,201	0,153
34	546	3,1	5,968	5,194
35	11	0,05	4,872	0,107
36	4	0,435	114,562	0,034
37	2	0,087	46,052	0,016
38	11	0,313	29,916	0,106
39	13	0,357	28,941	0,119
40	18	0,719	42,044	0,179
41	586	0,322	0,579	5,617
42	6	0,08	1,458	0,051
	4375	44,031	24,14028	41,78354

Как работает универсальный способ ускорения вычислений на этапе исполнения и снижает «алгоритмическую» емкость (длину исполняемого кода) при использовании пакетной обработки блоков данных? На диаграмме представлены 12 циклов обработки данных во времени и возможные способы их обработки более эффективным способом (1-ая диаграмма – описание значков диаграммы приведена на следующем слайде). Если процессор обладает хоть и ограниченной кэш-памятью, повторные итерации(!) будут исполняться из кэш-памяти еще быстрее (2-ая диаграмма).

	Циклы обработки												
Итерации	1	2	3	4	5	6	7	8	9	10	11	12	13
1		@		@		@		@		@		@	
2			@@		@@		@@		@@		@@		@@
3				@@@		@@@		@@@		@@@		@@@	
4					@@@@		@@@@		@@@@		@@@@		@@@@
5						@@@@@		@@@@@		@@@@@		@@@@@	
6							@@@@@@		@@@@@@		@@@@@@		@@@@@@
7								@@@@@@@		@@@@@@@		@@@@@@@	
8									@@@@@@@@		@@@@@@@@		@@@@@@@@
9										@@@@@@@@@		@@@@@@@@@	
10											@@@@@@@@@@	@@@@@@@@@@	

	Циклы обработки с усвоением из кэш-памяти(цикл обработки)												
Итерации	1	2	3	4	5	6	7	8	9	10	11	12	13
1		@		@		@		@		@		@	
2			@!		@!		@!		@!		@!		@!
3				@!!		@!!		@!!		@!!		@!!	
4					@!!!		@!!!		@!!!		@!!!		@!!!
5						@!!!!		@!!!!		@!!!!		@!!!!	
6							@!!!!!		@!!!!!		@!!!!!		@!!!!!
7								@!!!!!!		@!!!!!!		@!!!!!!	
8									@!!!!!!!		@!!!!!!!		@!!!!!!!
9										@!!!!!!!!		@!!!!!!!!	
10											@!!!!!!!!!	@!!!!!!!!!	

Работающие технические аналоги предлагаемой технологии:

- Чтобы сократить влияние фактора латентности оперативной памяти, процессор использует специальную технологию, именуемую пакетно-монопольным (групповым или просто монопольным) режимом (burst mode) – каждый раз читает или записывает по 8 блоков памяти – по кэш строке (64 байта);
- При обмене с жесткими дисками операционная система читает полностью кластер или сектор, а потом отдает нужный блок потребителю.

• Теория и расчетные соотношения (плоская модель накладных расходов):

N – количество одиночных блоков данных, которые необходимо обработать для решения задачи;

k – размер пакета обрабатываемых блоков информации;

$L = N/k$ – количество подключений процессора для обработки N блоков по одному или пакетами;

n – количество накладных команд процессора при каждом подключении процессора ( – зеленый прямоугольник на диаграмме);

c – цикл обработки в командах процессора одинарного блока, который может быть выполнен как однократно, так и несколько раз при подключении задачи (@...@@@ - на диаграмме);

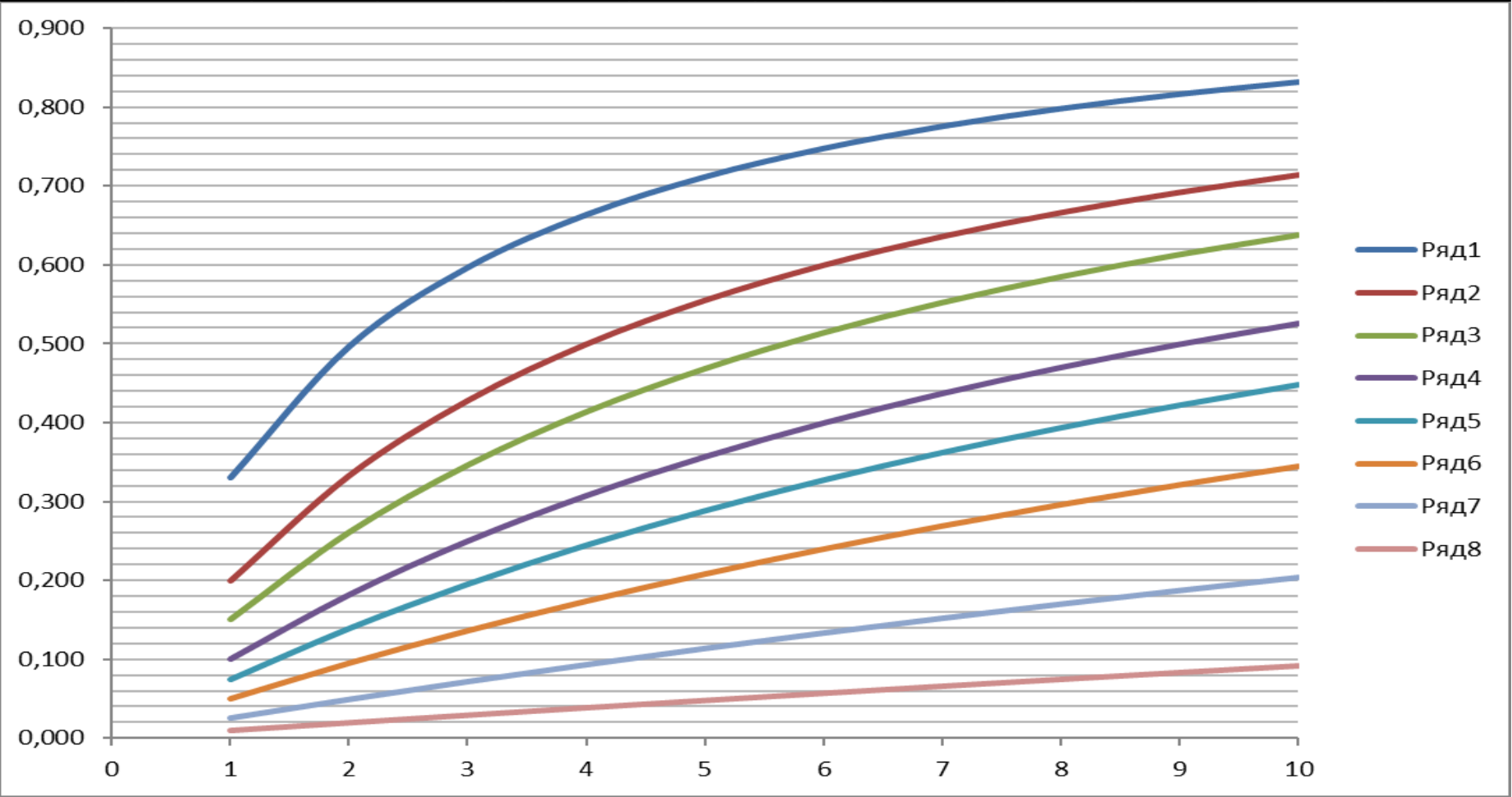
$a = c/(n+c)$ – исходная продуктивность кода;

$a_k = ka/(1+a(k-1))$ – продуктивность кода при пакете обработки в k блоков;

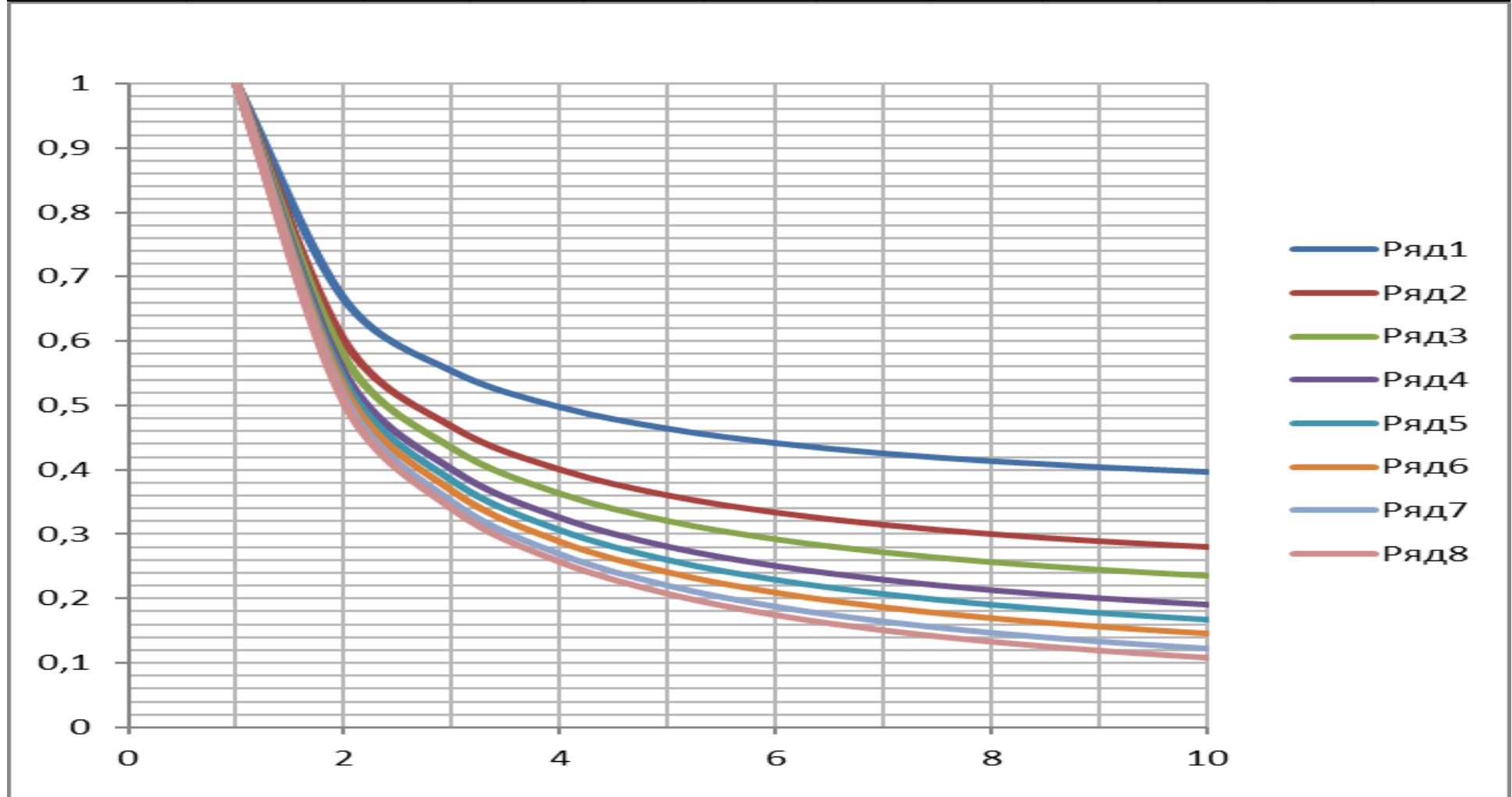
$s_k = a+(1-a)/k$ – нормированная длина кода при пакете обработки в k блоков;

$p_k = ((1-a)/k+a)/k$ – нормированная загрузка процессора обработкой пакета в k блоков.

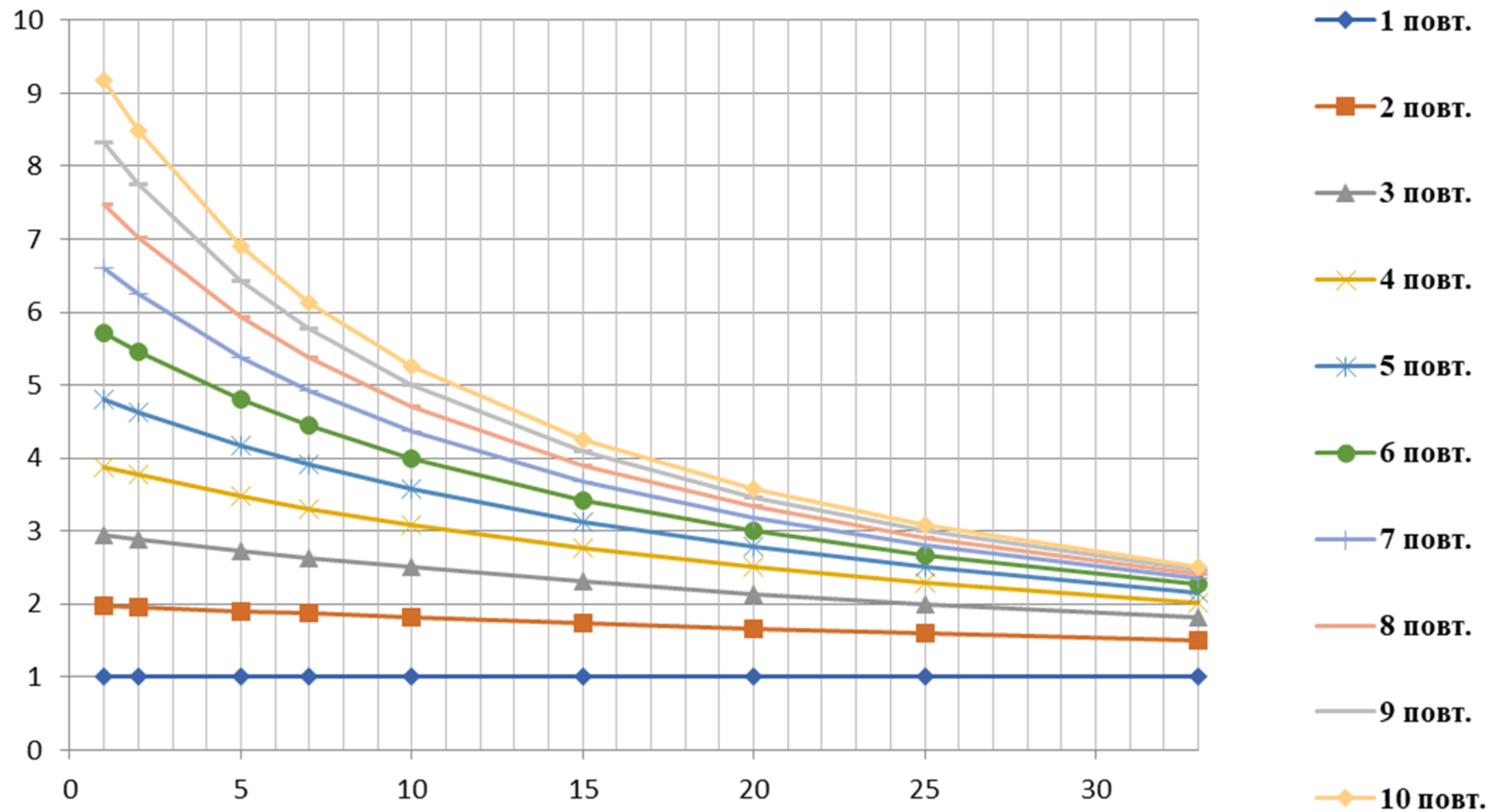
Продуктивность от рамера пакета обрабатываемых данных											
№ п.п.	Исходная	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
1	0,330	0,330	0,496	0,596	0,663	0,711	0,747	0,775	0,798	0,816	0,831
2	0,200	0,200	0,333	0,429	0,500	0,556	0,600	0,636	0,667	0,692	0,714
3	0,150	0,150	0,261	0,346	0,414	0,469	0,514	0,553	0,585	0,614	0,638
4	0,100	0,100	0,182	0,250	0,308	0,357	0,400	0,438	0,471	0,500	0,526
5	0,075	0,075	0,140	0,196	0,245	0,288	0,327	0,362	0,393	0,422	0,448
6	0,050	0,050	0,095	0,136	0,174	0,208	0,240	0,269	0,296	0,321	0,345
7	0,025	0,025	0,049	0,071	0,093	0,114	0,133	0,152	0,170	0,188	0,204
8	0,010	0,010	0,020	0,029	0,039	0,048	0,057	0,066	0,075	0,083	0,092



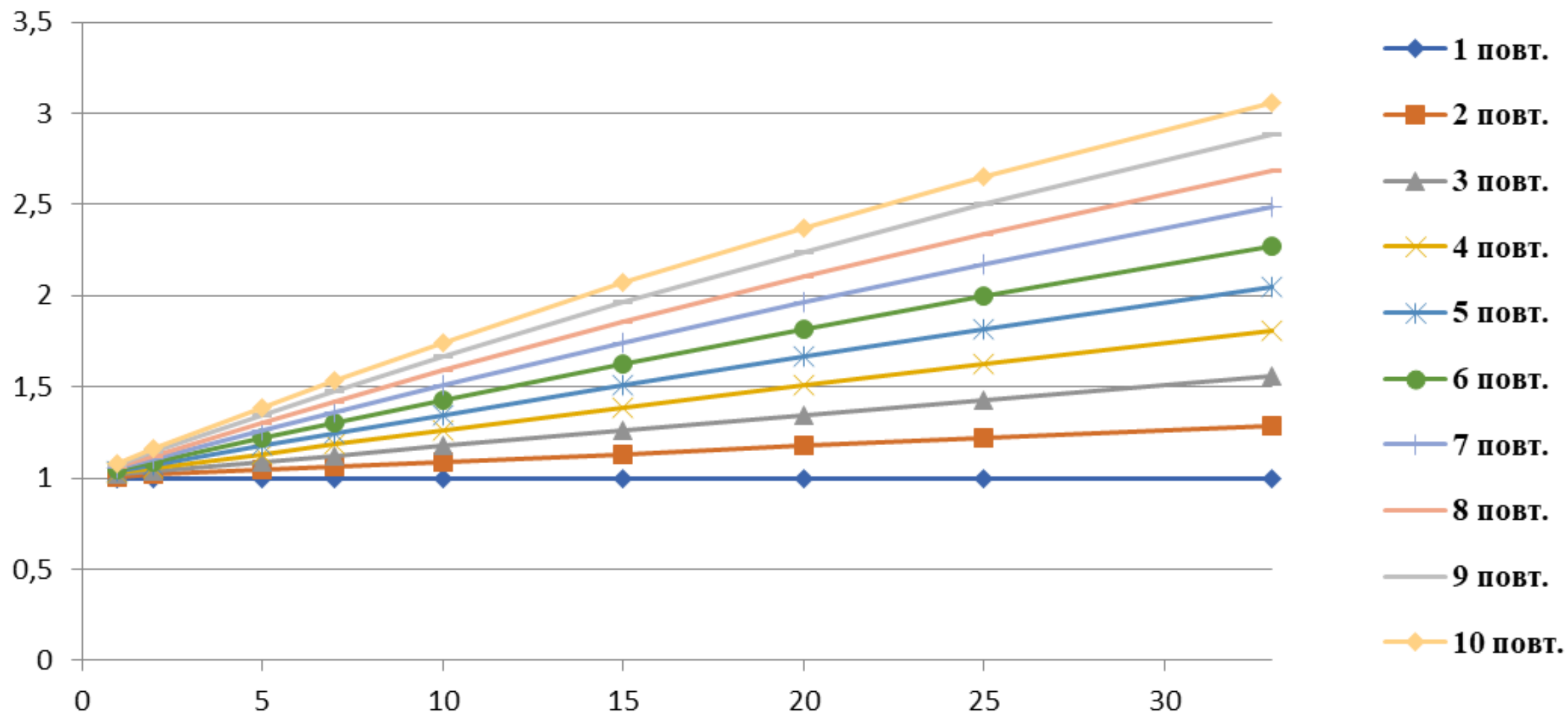
Нормированная длина кода от рамера пакета обрабатываемых данных											
№ п.п.	Исходная	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
1	0,330	1,000	0,665	0,553	0,498	0,464	0,442	0,426	0,414	0,404	0,397
2	0,200	1,000	0,600	0,467	0,400	0,360	0,333	0,314	0,300	0,289	0,280
3	0,150	1,000	0,575	0,433	0,363	0,320	0,292	0,271	0,256	0,244	0,235
4	0,100	1,000	0,550	0,400	0,325	0,280	0,250	0,229	0,213	0,200	0,190
5	0,075	1,000	0,538	0,383	0,306	0,260	0,229	0,207	0,191	0,178	0,168
6	0,050	1,000	0,525	0,367	0,288	0,240	0,208	0,186	0,169	0,156	0,145
7	0,025	1,000	0,513	0,350	0,269	0,220	0,188	0,164	0,147	0,133	0,123
8	0,010	1,000	0,505	0,340	0,258	0,208	0,175	0,151	0,134	0,120	0,109



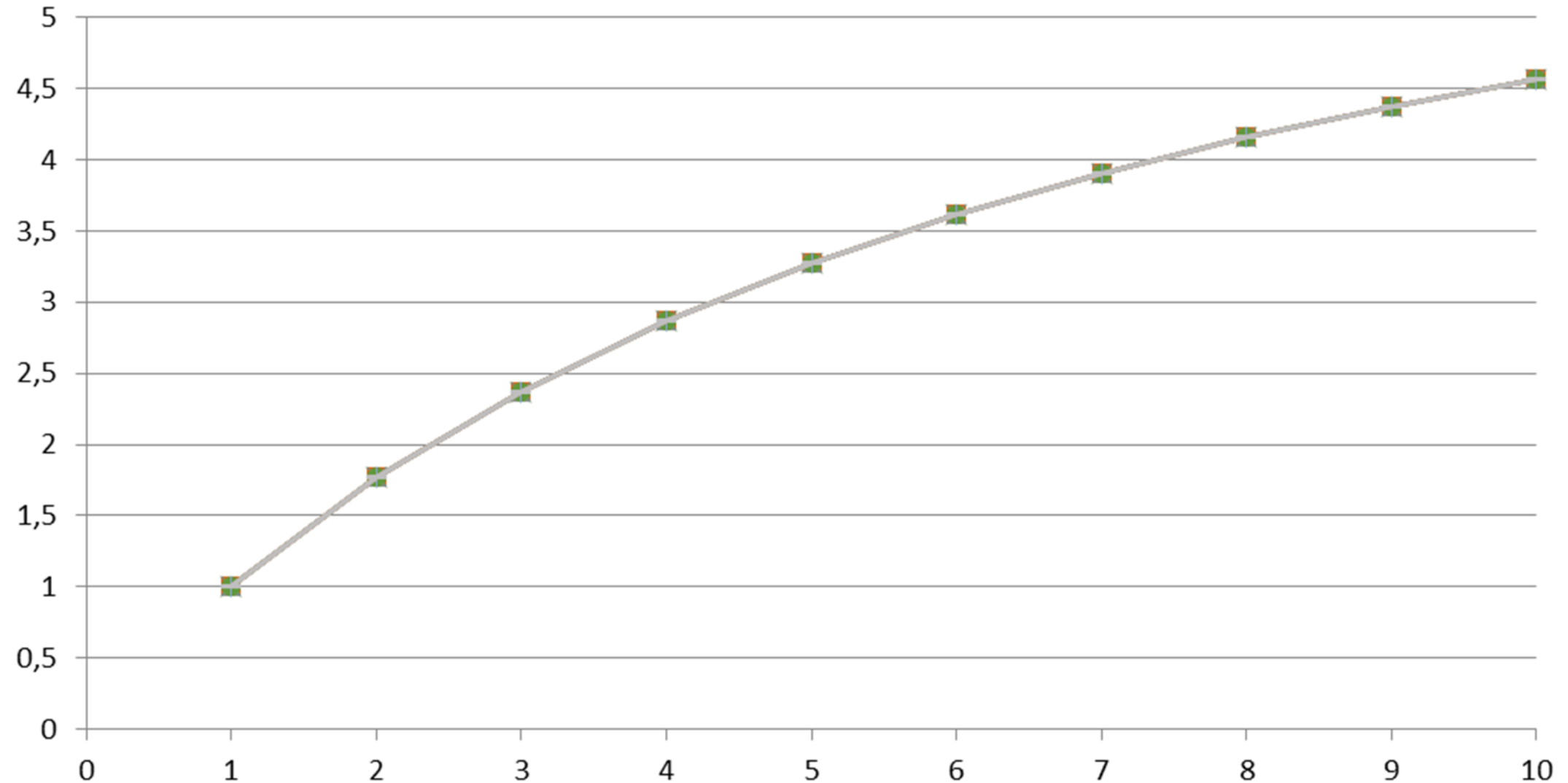
Возможности сокращения длины кода в размах в зависимости от продуктивности кода в % на этапе исполнения и количества повторов

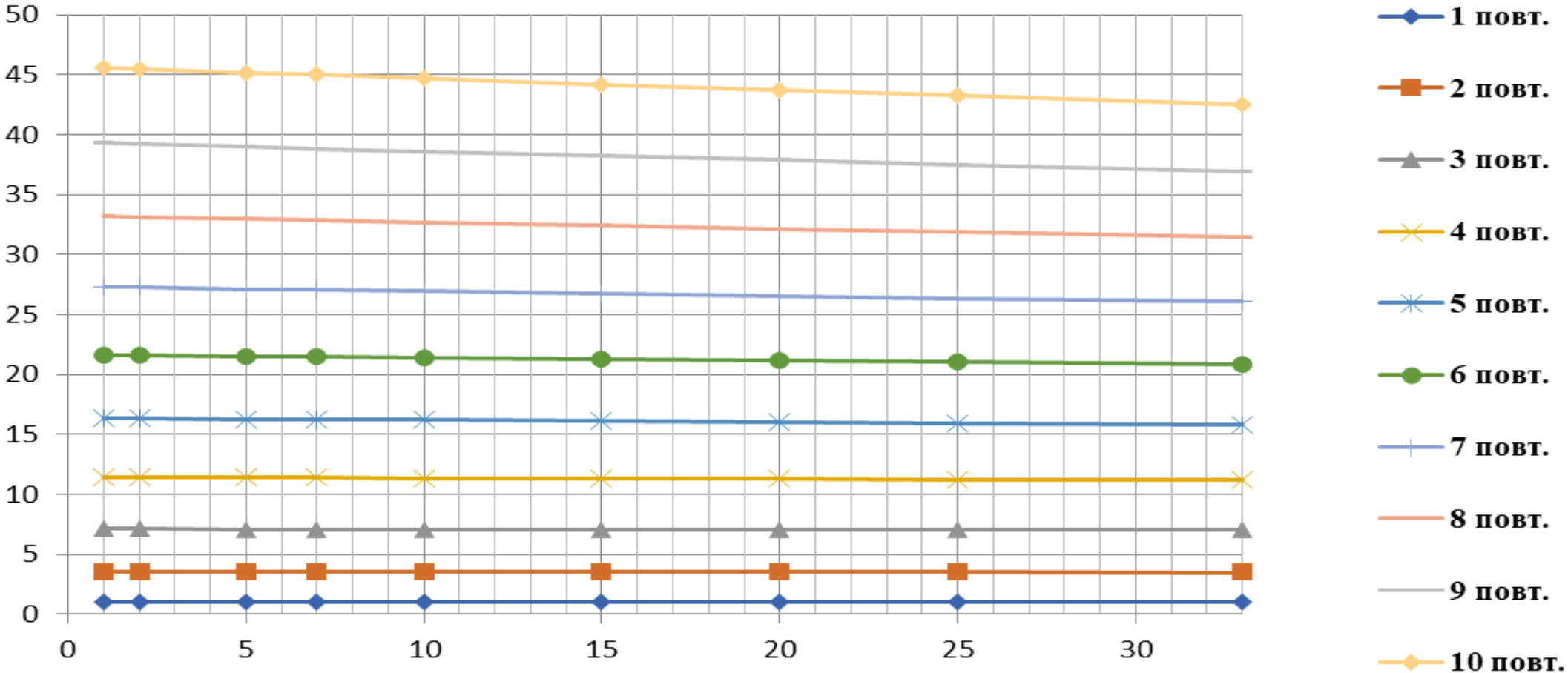


Увеличение скорости исполнения целевого кода при повторении из кэш-памяти от количества повторений в разях



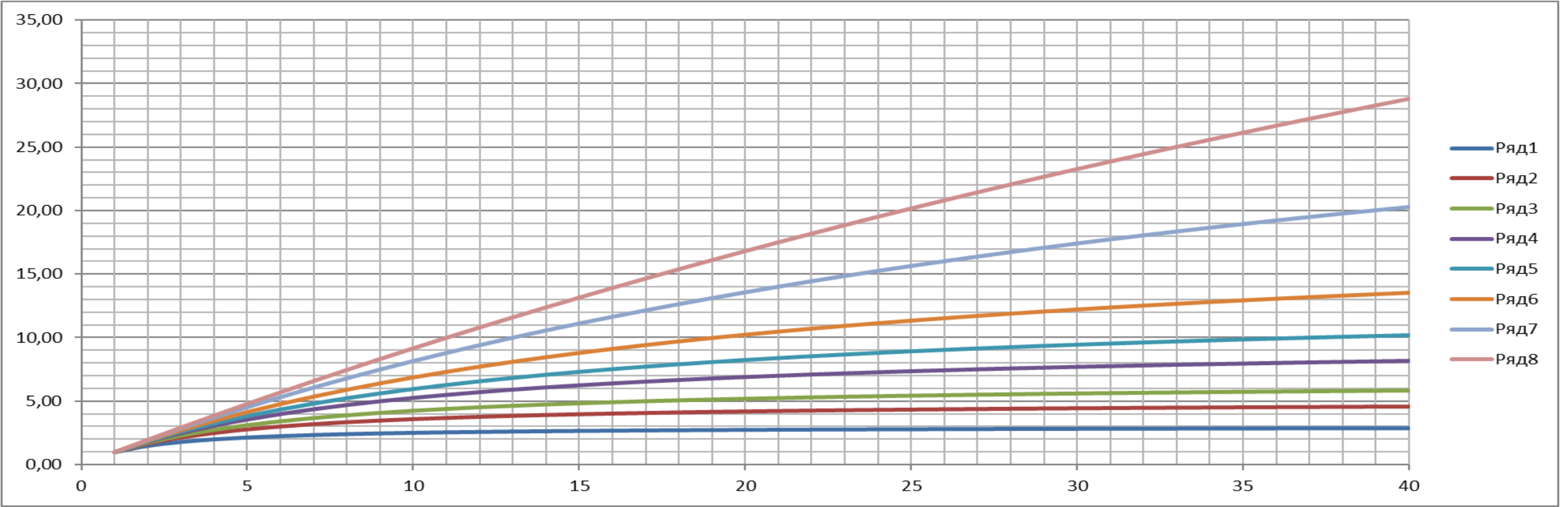
Уменьшение загрузки шины памяти чтением кода и данных в размах при повторах от 2 до 10





Вот так будет выглядеть суммарное увеличение производительности в зависимости от размера пакета обрабатываемых данных до k=40(это для режима ядра ОС – верхние две эпюры для продуктивности 1% и 2.5% на нижнем рисунке) от 20 до 29 раз (только за счет сокращения длины исполняемого кода)! Поскольку при такой низкой продуктивности кэш практически не добавляет скорости суммарное ускорение с учетом разгрузки памяти (x4,5)составит от 90 до 130 раз! На 2 порядка можно увеличить быстродействие ядра!

№ п.п.	Исходная	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14	k=15	k=16	k=17	k=18	k=19	k=20	k=21	k=22	k=23	k=24	k=25	k=26	k=27	k=28	k=29	k=30	k=31	k=32	k=33	k=34	k=35	k=36	k=37	k=38	k=39	k=40	
1	0,330	1,000	0,665	0,553	0,498	0,464	0,442	0,426	0,414	0,404	0,397	0,391	0,386	0,382	0,378	0,375	0,372	0,369	0,367	0,365	0,364	0,362	0,360	0,359	0,358	0,357	0,356	0,355	0,354	0,353	0,352	0,352	0,351	0,350	0,350	0,349	0,349	0,348	0,348	0,347	0,347	
2	0,200	1,000	0,600	0,467	0,400	0,360	0,333	0,314	0,300	0,289	0,280	0,273	0,267	0,262	0,257	0,253	0,250	0,247	0,244	0,242	0,240	0,238	0,236	0,235	0,233	0,232	0,231	0,230	0,229	0,228	0,227	0,226	0,225	0,224	0,224	0,223	0,222	0,222	0,221	0,221	0,220	0,220
3	0,150	1,000	0,575	0,433	0,363	0,320	0,292	0,271	0,256	0,244	0,235	0,227	0,221	0,215	0,211	0,207	0,203	0,200	0,197	0,195	0,193	0,190	0,189	0,187	0,185	0,184	0,183	0,181	0,180	0,179	0,178	0,177	0,177	0,176	0,175	0,174	0,174	0,173	0,172	0,172	0,171	0,171
4	0,100	1,000	0,550	0,400	0,325	0,280	0,250	0,229	0,213	0,200	0,190	0,182	0,175	0,169	0,164	0,160	0,156	0,153	0,150	0,147	0,145	0,143	0,141	0,139	0,138	0,136	0,135	0,133	0,132	0,131	0,130	0,129	0,128	0,127	0,126	0,126	0,125	0,124	0,124	0,123	0,123	0,123
5	0,075	1,000	0,538	0,383	0,306	0,260	0,229	0,207	0,191	0,178	0,168	0,159	0,152	0,146	0,141	0,137	0,133	0,129	0,126	0,124	0,121	0,119	0,117	0,115	0,114	0,112	0,111	0,109	0,108	0,107	0,106	0,105	0,104	0,103	0,102	0,101	0,101	0,100	0,099	0,099	0,098	0,098
6	0,050	1,000	0,525	0,367	0,288	0,240	0,208	0,186	0,169	0,156	0,145	0,136	0,129	0,123	0,118	0,113	0,109	0,106	0,103	0,100	0,098	0,095	0,093	0,091	0,090	0,088	0,087	0,085	0,084	0,083	0,082	0,081	0,080	0,079	0,078	0,077	0,076	0,076	0,075	0,074	0,074	0,074
7	0,025	1,000	0,513	0,350	0,269	0,220	0,188	0,164	0,147	0,133	0,123	0,114	0,106	0,100	0,095	0,090	0,086	0,082	0,079	0,076	0,074	0,071	0,069	0,067	0,066	0,064	0,063	0,061	0,060	0,059	0,058	0,056	0,055	0,055	0,054	0,053	0,052	0,051	0,051	0,050	0,049	0,049
8	0,010	1,000	0,505	0,340	0,258	0,208	0,175	0,151	0,134	0,120	0,109	0,100	0,093	0,086	0,081	0,076	0,072	0,068	0,065	0,062	0,060	0,057	0,055	0,053	0,051	0,050	0,048	0,047	0,045	0,044	0,043	0,042	0,041	0,040	0,039	0,038	0,038	0,037	0,036	0,035	0,035	0,035



Средняя статистика кода процессоров X86_64 полученных на разработанном мониторе-профилировщике Speedprograf собственной разработки

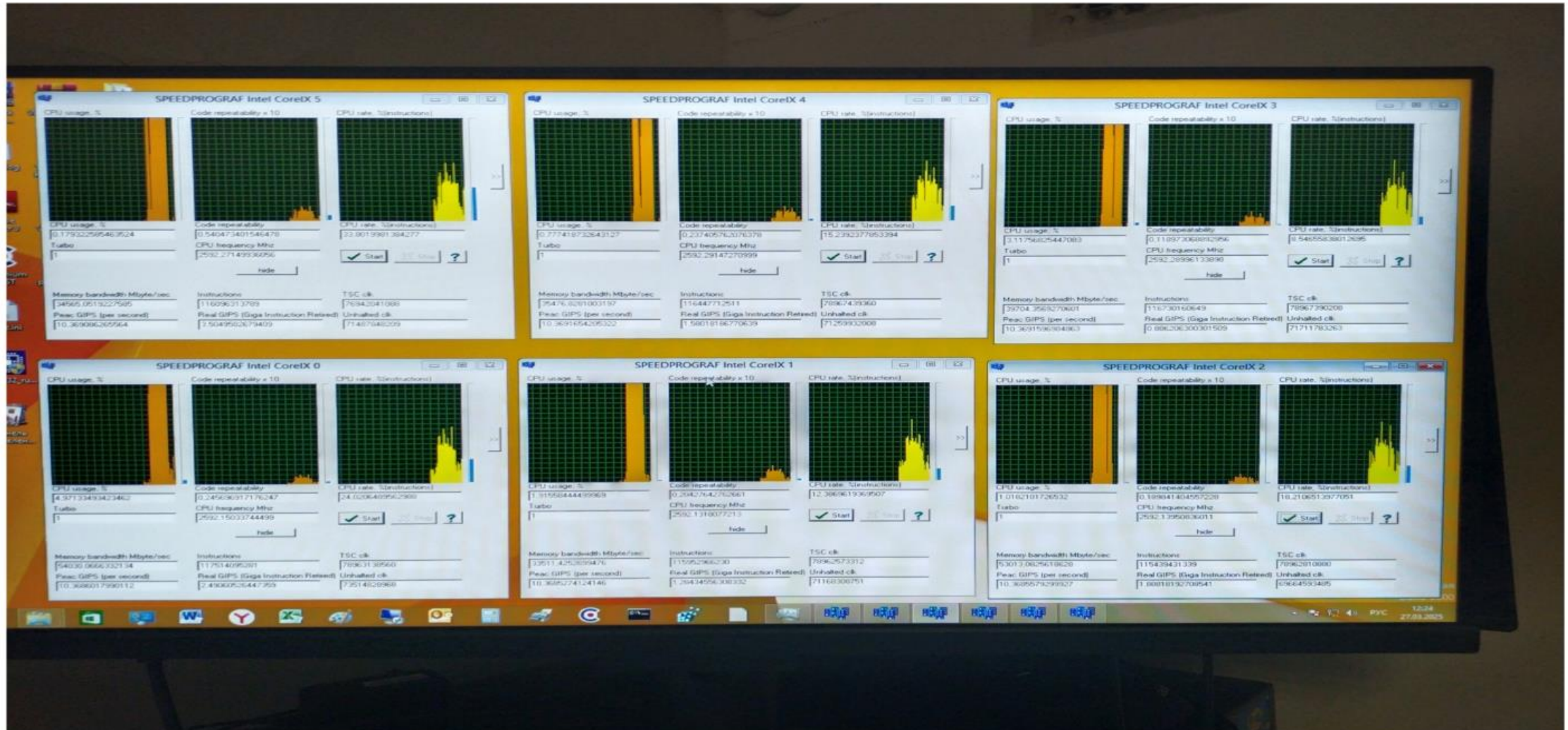
Показатели сравнения	X86	X64
Разрядность, бит	32	64
Доля команд управления	1/5	1/5
Доля команд обращения к памяти	1/4	1/4
Доля команд чтения данных	1/6	1/6
Доля команд записи данных	1/12	1/12
Средняя длина команды байт	5	6
Доля данных в обращении к памяти	1/5	1/3

Это процессор Intel Core2 Quad Q9450 2.66Ghz на памяти DDR2-6400 (12800 -Dual mode) с ATA-SSD и с кэшем L2 12 Мбайт демонстрирует 24%пиковой производительности 4-х одновременных архиваций WinRar «скоростной» 4-х ядер процессора при превышении суммарного потока кода и данных 54400 Мбайт в секунду (Code repeatability - в 0,9-1.51 раза поток каждого процессора) и превышает пиковую пропускную способность двухканальной памяти в 4,25 раза (точный суммарный расчет по счетчикам MSR), т.е. кэш ускоряет вычисления в 4,25 раз!

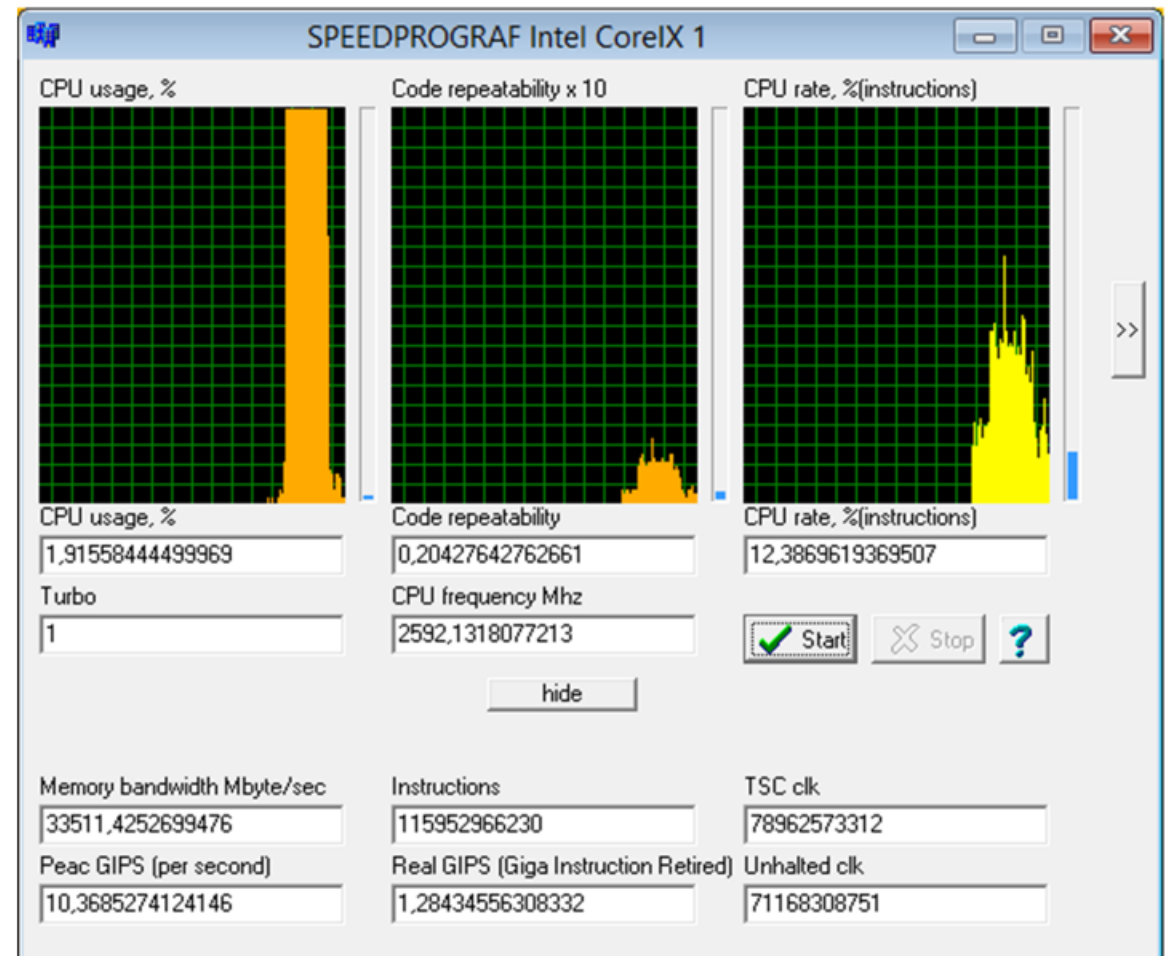
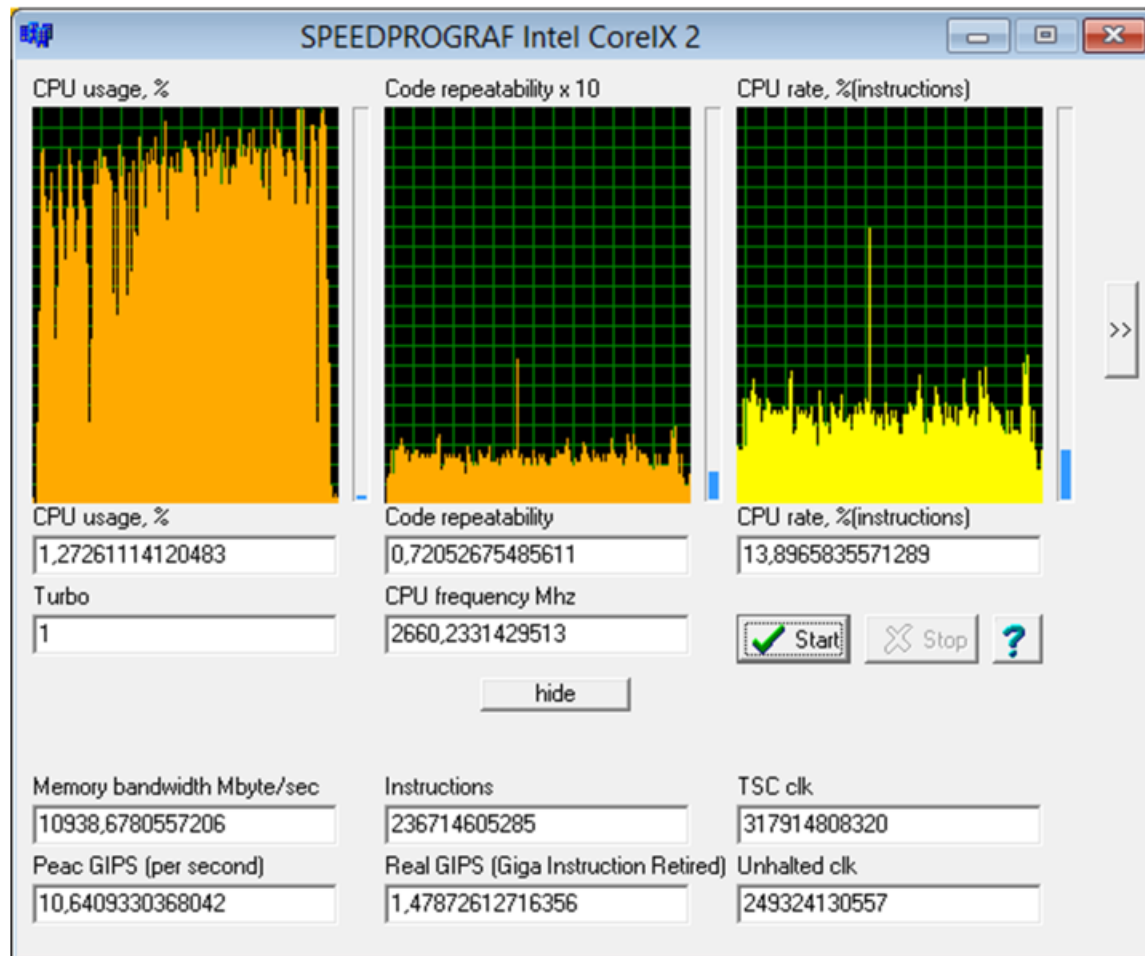


Здесь использован диск SSD.M2(Nvme). Средний график Code repeatability (повторяемость код в разях) показывает, во сколько раз код выполняется быстрее установленной памяти. В данном случае она не превышает 1, видно что все ядра уперлись в память (загрузка каждого 100%). Структурные конфликты не дают работать быстрее(ниже будет подробнее про них).

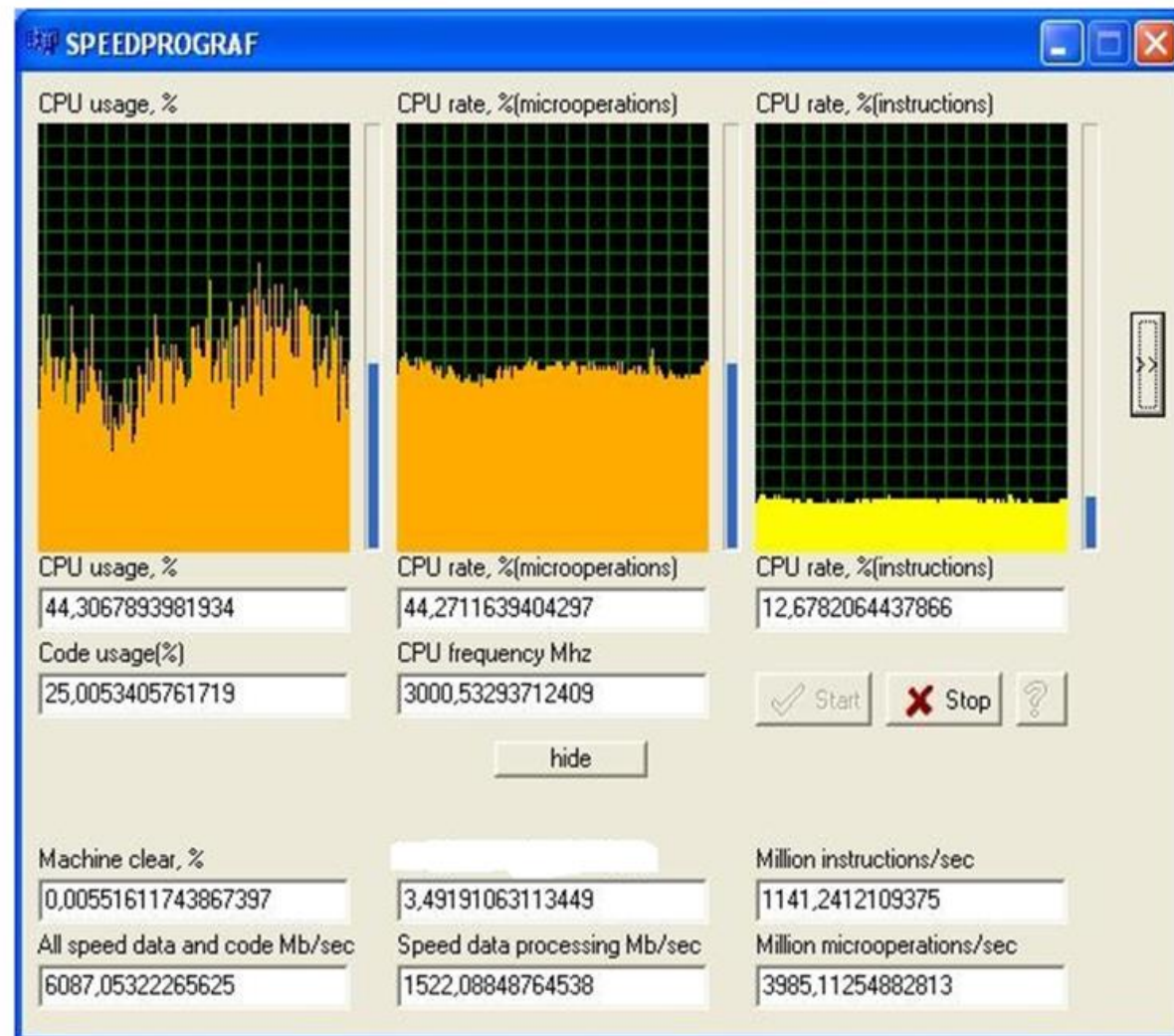
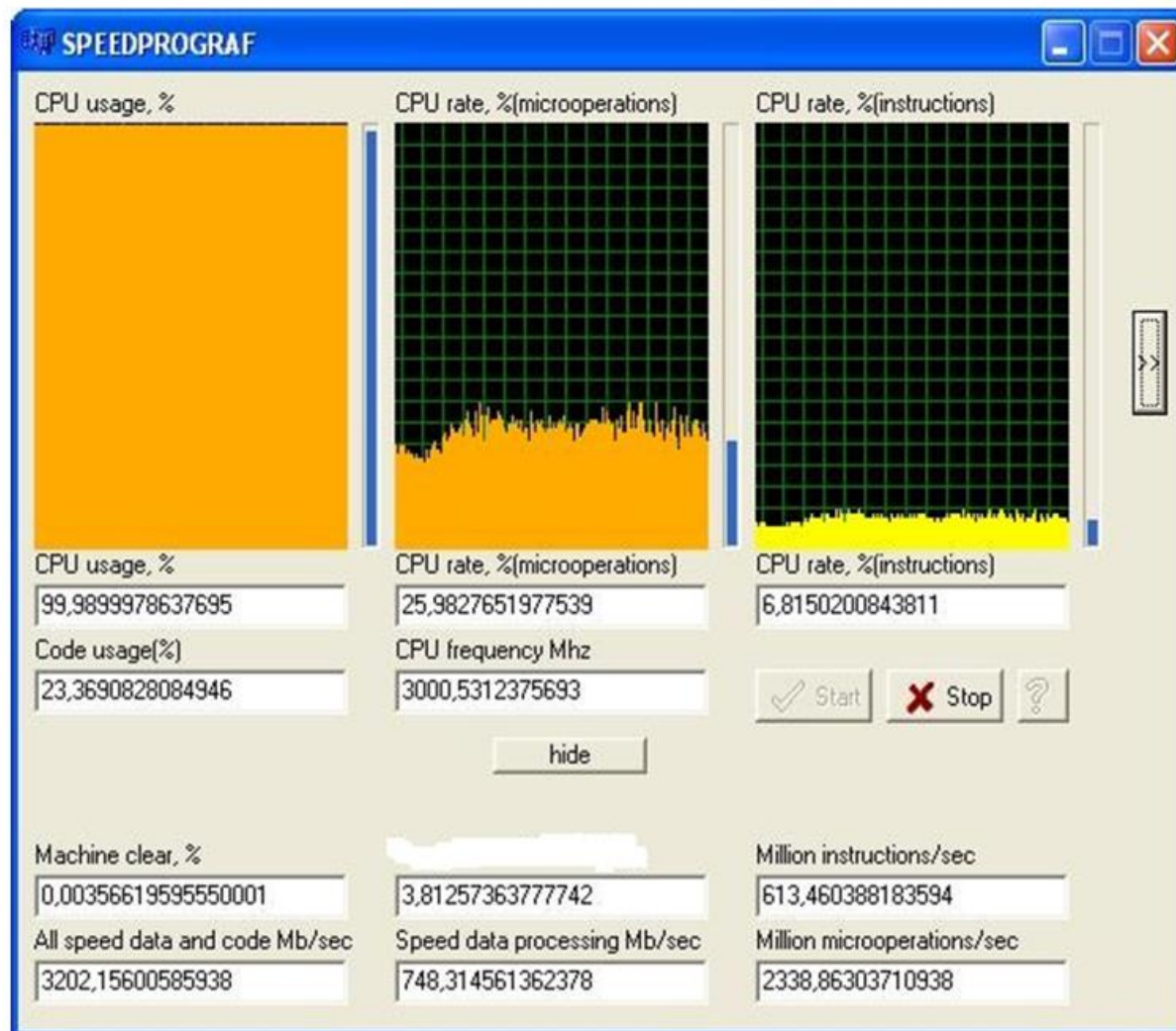
Это процессор Intel Corei5 11400F 2.6Ghz на памяти DDR4-25600 (51200 -Dual mode) с M2-SSD и с кэшем L3 12 Мбайт демонстрирует 40,6 % пиковой производительности на 6-ти-одновременных архиваций WinRar «скоростной» 6-ти ядер процессора при превышении суммарного потока кода и данных 124240 Мбайт в секунду (Code repeatability - в 0,7-1,2 раза каждый процессор) и превышает пиковую пропускную способность двухканальной памяти в 2,43 раза (точный суммарный расчет по счетчикам MSR), т.е. кэш ускоряет вычисления в 2,43 раза!



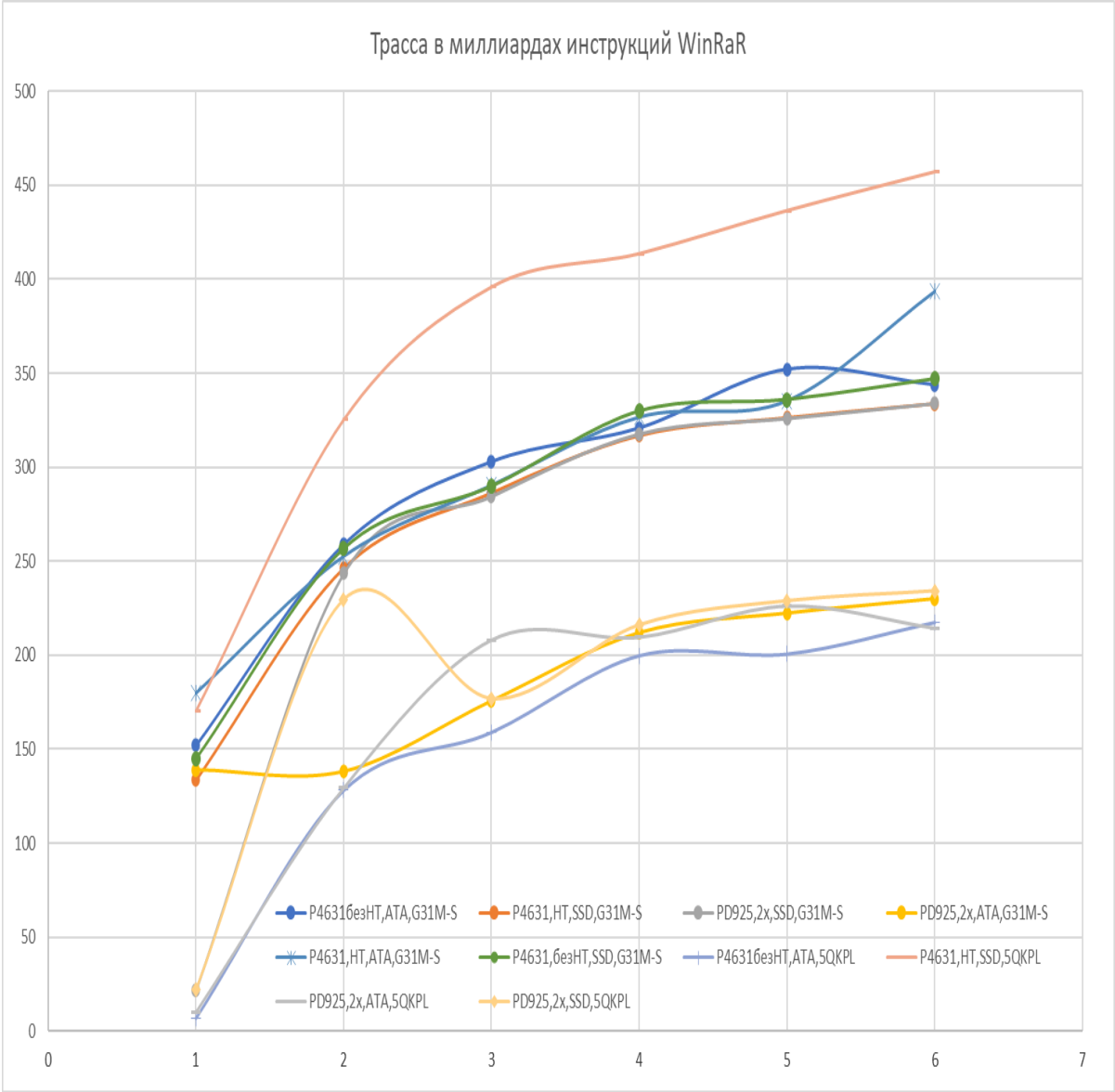
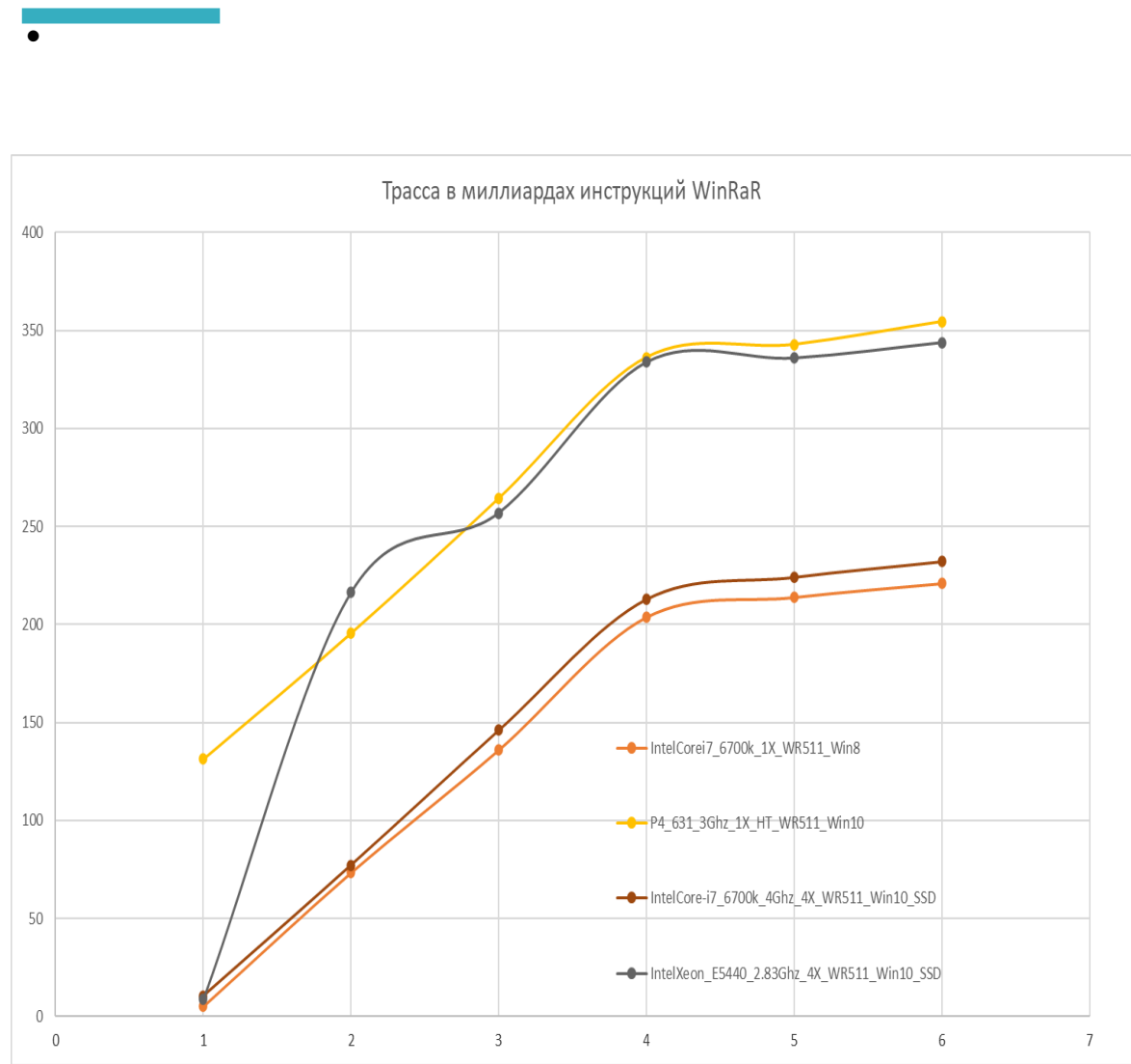
Теперь обратим внимание на окошко «Instructions», которое показывает какой путь прошел процессор при выполнении данного задания или сколько процессорных команд было потрачено. Архиватор один и тот же, WinRar 6.23, но у Intel core2 Quad q9450 236,7 миллиардов инструкций, а у Intel Core I5 11400F – 115,95 миллиардов? В 2,04 раза меньше и время меньше раз в шесть, тут всем все понятно? Там диски SSD отличаются по скорости на порядок, потом время выполнения сильно отличается.



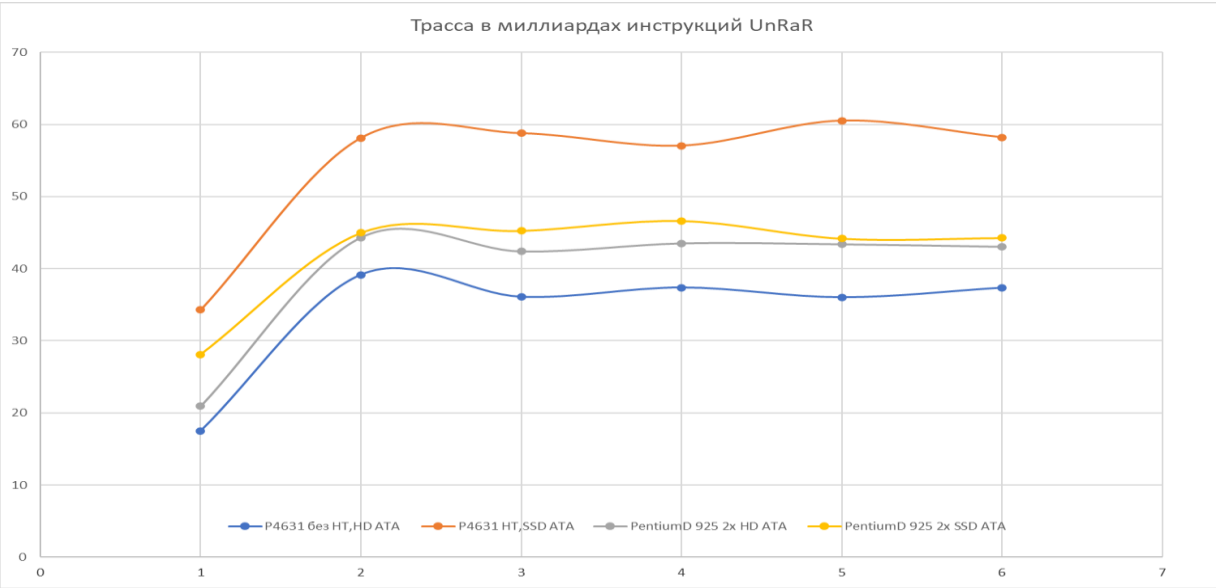
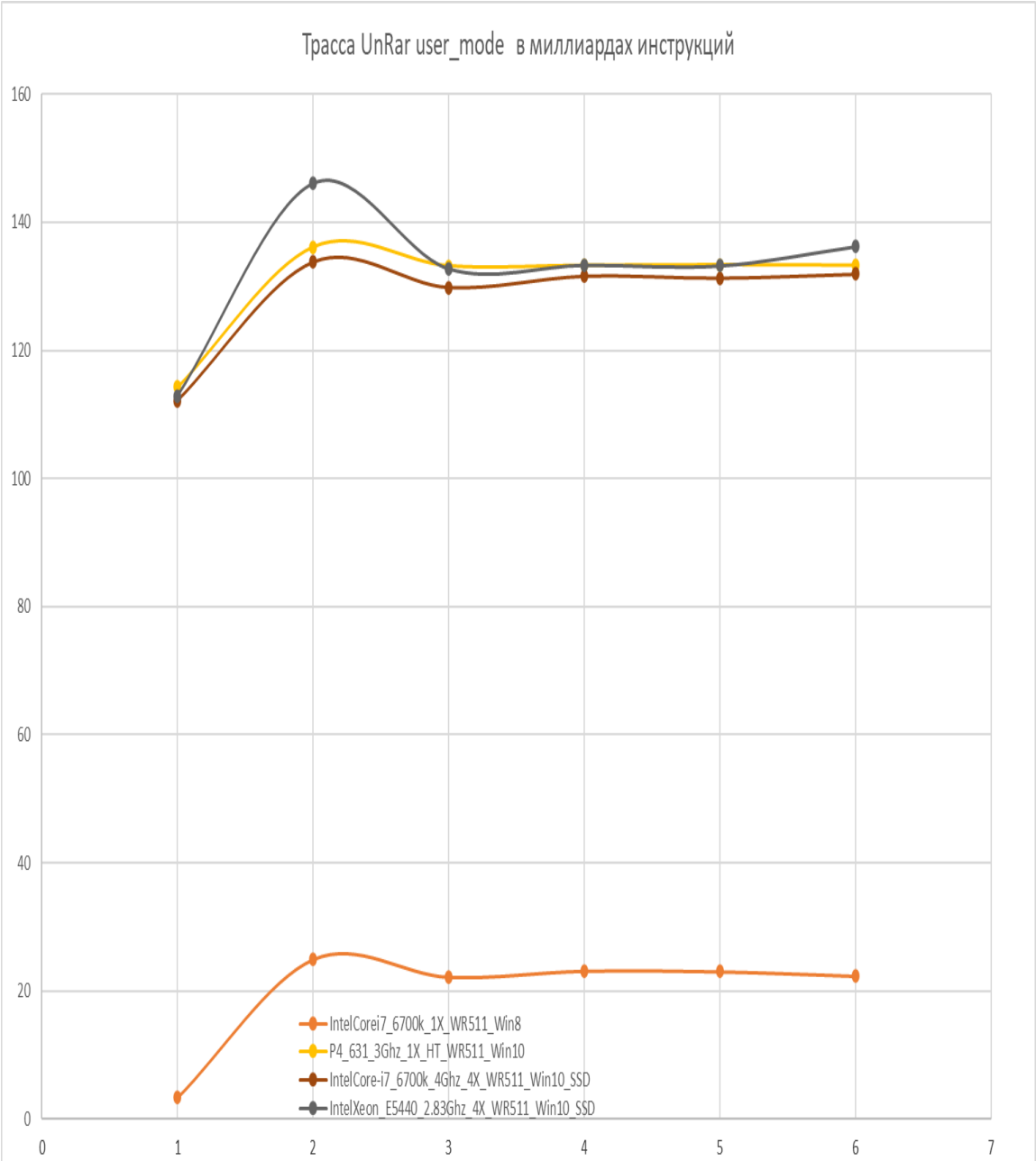
Демонстрация увеличения производительности путем сокращения трассы за счет увеличения повторяемости кода на одном процессоре Intel PIV в одинаковых условиях. Это продолжение сокращения кода на том же приложении но на другой платформе со 111 миллиардов инструкций до 54 миллиардов еще в два раза короче! Загрузка процессора снизилась в 2 раза (трасса короче больше чем в 2 раза), скорость обработки выросла в 2 раза (с 6,81% до 12,68% (еще в 8 раз разогнать можно)), а время архивации сократилось в 4 раза (квадратичная зависимость!). Это демонстрация частичного ускорения в действии! Исходная продуктивность 11,5%.!! От первой длины на предыдущем рисунке мы «сократили» длину исполняемого кода ($236,7/54=4,38$) в 4 раза! И это не предел!



Различие трасс современных процессоров на архивацию WinRaR одного и того-же массива данных по регистрации профилировщика Speedprograf (архивы совпадают до байта)? Одна и та же работа может быть выполнена разными путями? И можно-ли сократить трассу – ответ положительный! Мой профилировщик работает на любых платформах от Intel и AMD, от первых до последних семейств(данные приведены от разных платформ, а не устаревшие технологии). И разные платформы показывают одинаковые уровни трасс разного порядка. (на левом рисунке 2 и 3 порядка, на правом –1,2 и 3 порядка)

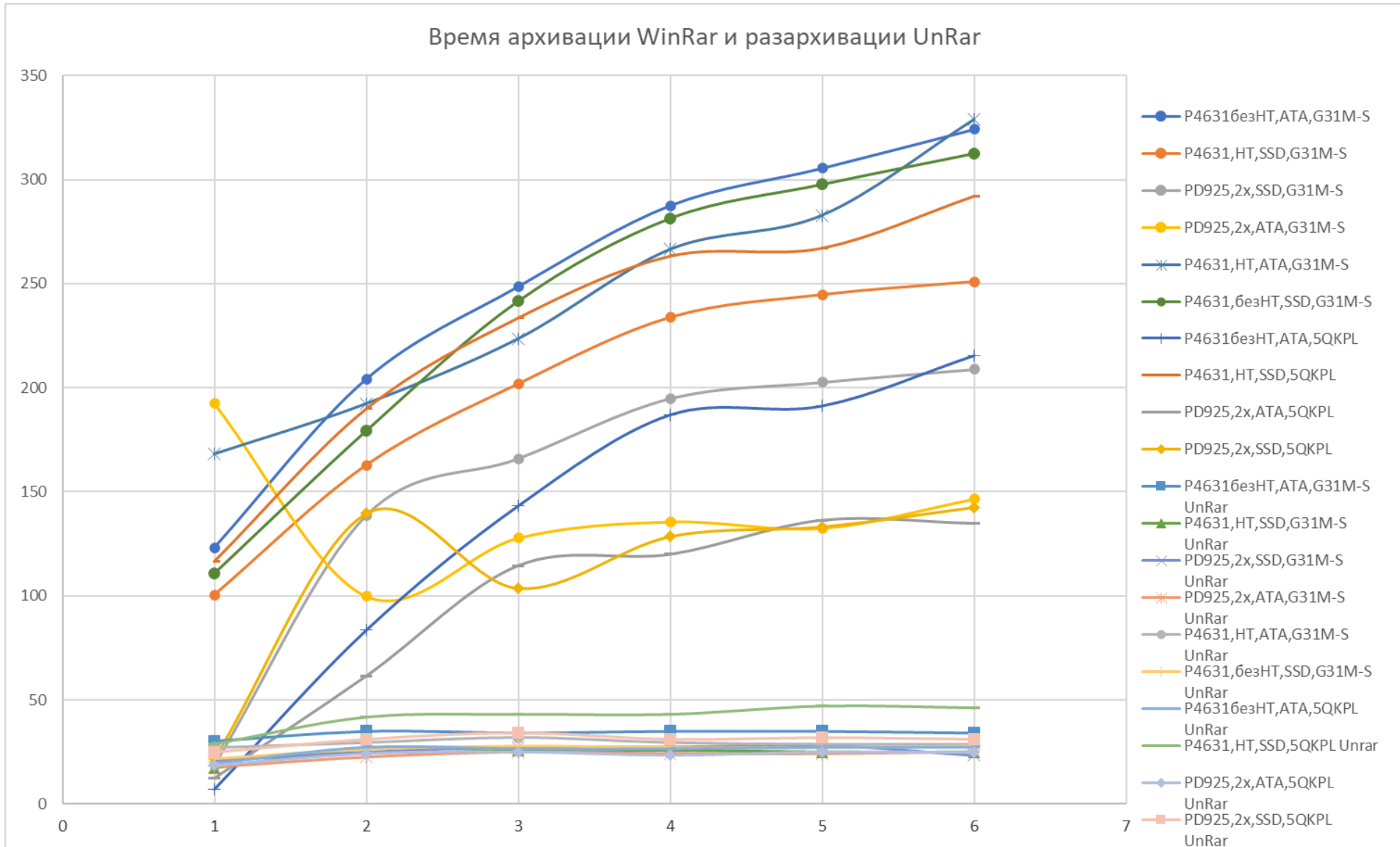


Трассы разархивации UnRAR в одном масштабе различных платформ, отличающихся реальным быстродействием. Реально быстродействующие платформы больше тратят накладных расходов на одну и ту-же работу.

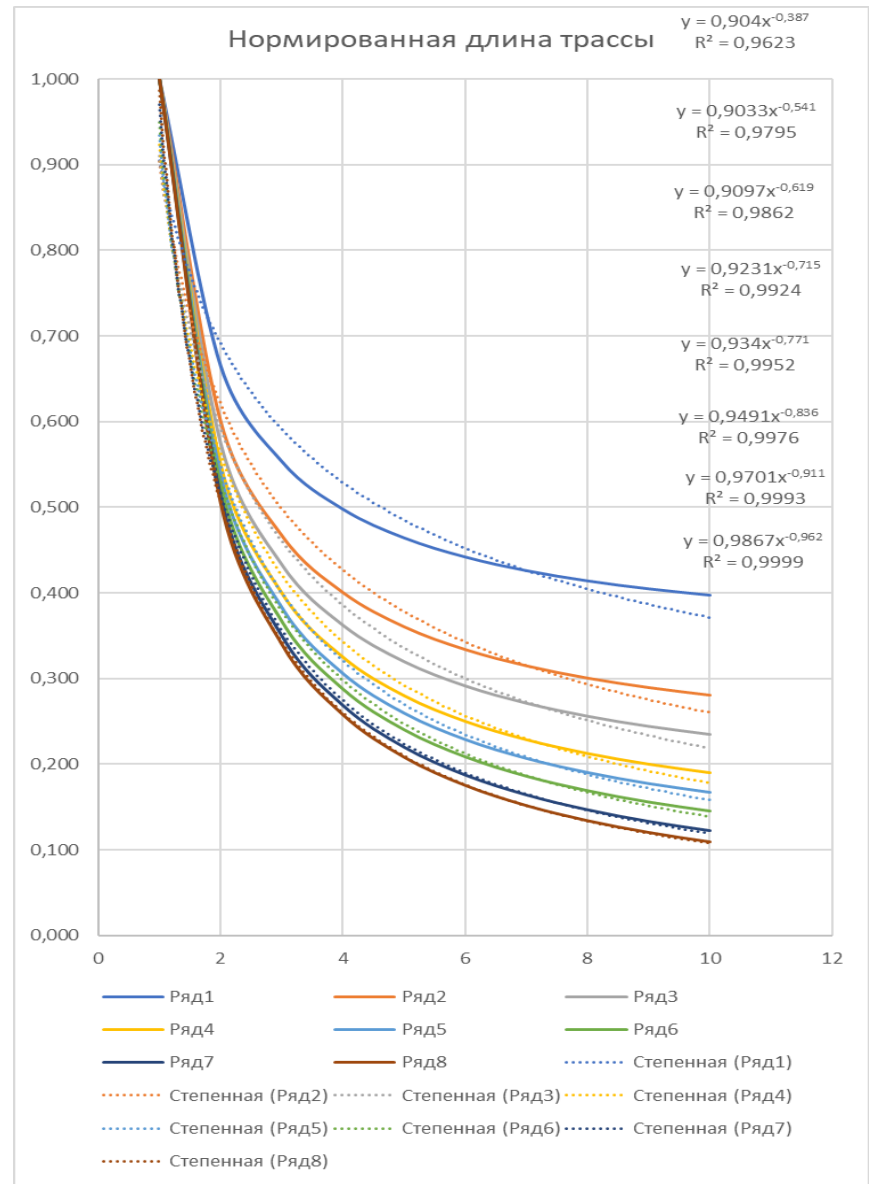




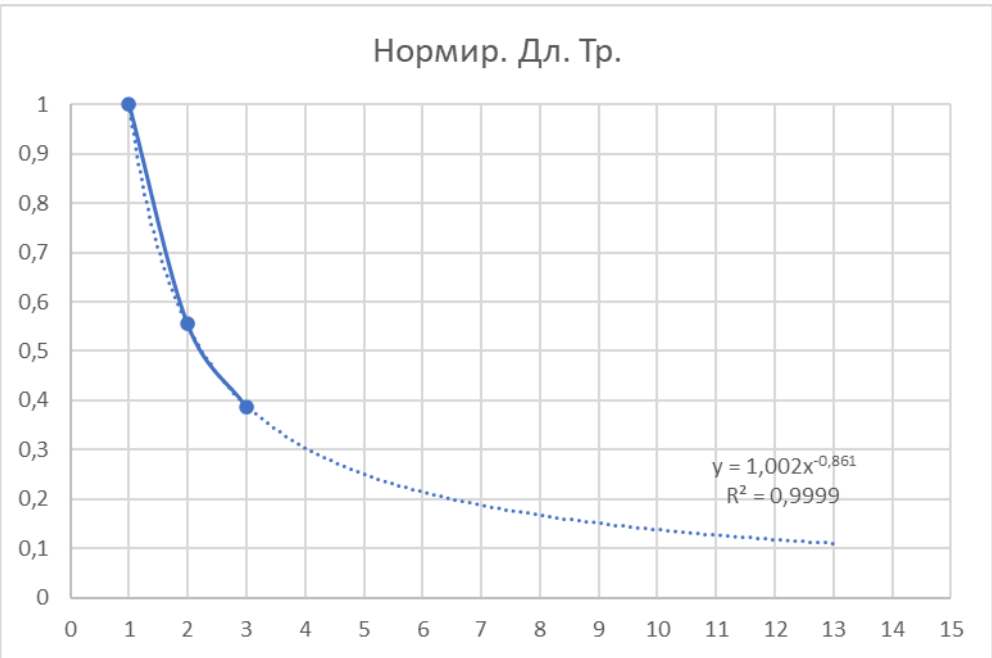
Что дает блочная обработка? Ускорение больше, чем больше блок!!! В разархивации блок на 1 больше соответственно всегда! Не 1-2-3, а соответственно 2-3-4!!



Степенная модель(минимальная точность модели при исходной продуктивности кода <0,15(15%), максимальная–1-2%). Продуктивность кода. Пример расчета по трем точкам в нижней части слайда.

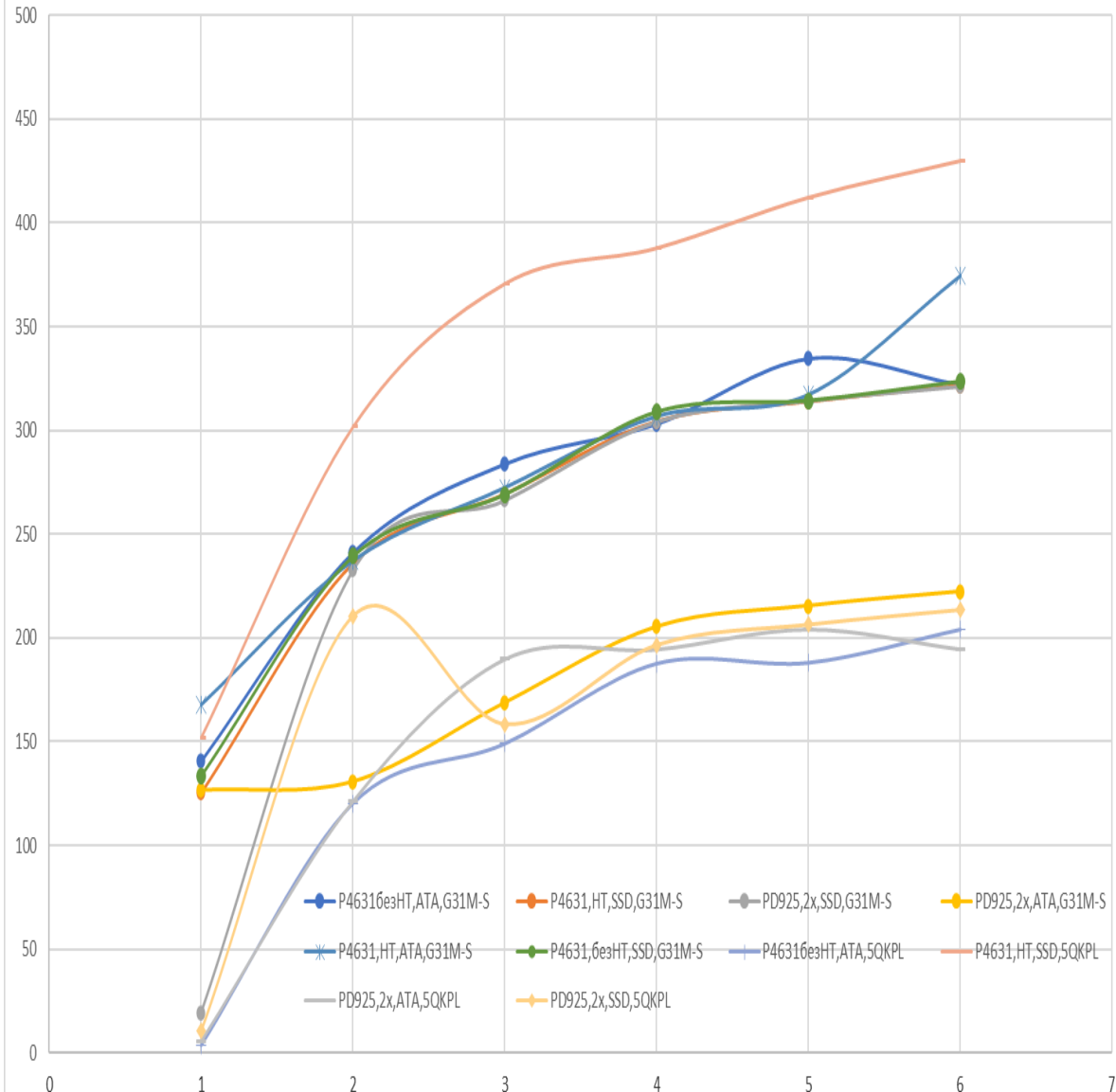


Нормированная длина кода от размера пакета обрабатываемых данных												
№ п.п.	Исходная	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	
1	0,330	1,000	0,665	0,553	0,498	0,464	0,442	0,426	0,414	0,404	0,397	
2	0,200	1,000	0,600	0,467	0,400	0,360	0,333	0,314	0,300	0,289	0,280	
3	0,150	1,000	0,575	0,433	0,363	0,320	0,292	0,271	0,256	0,244	0,235	
4	0,100	1,000	0,550	0,400	0,325	0,280	0,250	0,229	0,213	0,200	0,190	
5	0,075	1,000	0,538	0,383	0,306	0,260	0,229	0,207	0,191	0,178	0,168	
6	0,050	1,000	0,525	0,367	0,288	0,240	0,208	0,186	0,169	0,156	0,145	
7	0,025	1,000	0,513	0,350	0,269	0,220	0,188	0,164	0,147	0,133	0,123	
8	0,010	1,000	0,505	0,340	0,258	0,208	0,175	0,151	0,134	0,120	0,109	

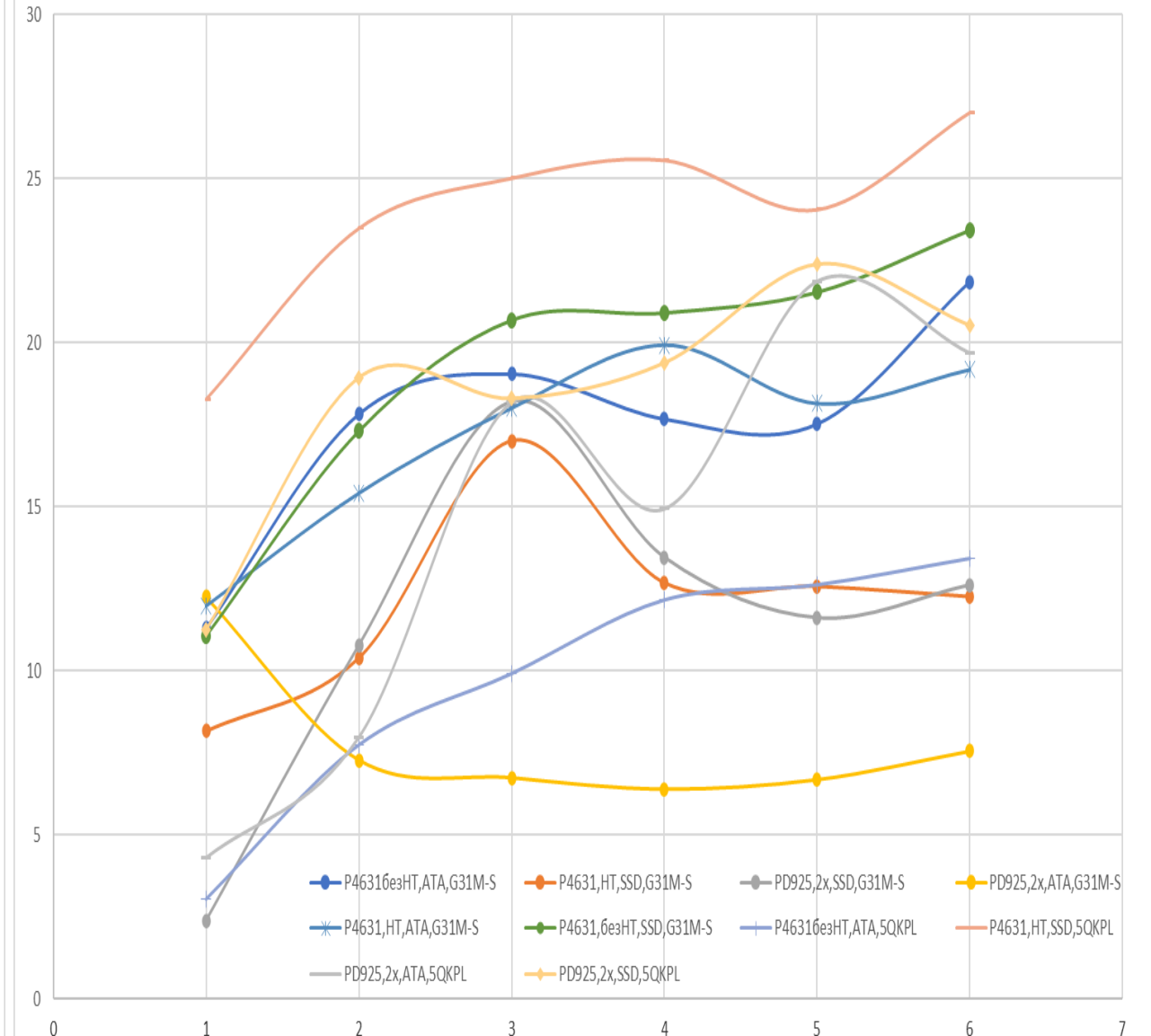


Расчет реальной продуктивности отдельно по user_mode и os_mode (для user_mode выбирается 3 усредненных уровня(с 1 по 3), для os_mode 4 уровня (с 1 по 4))

Трасса WinRAR user_mode



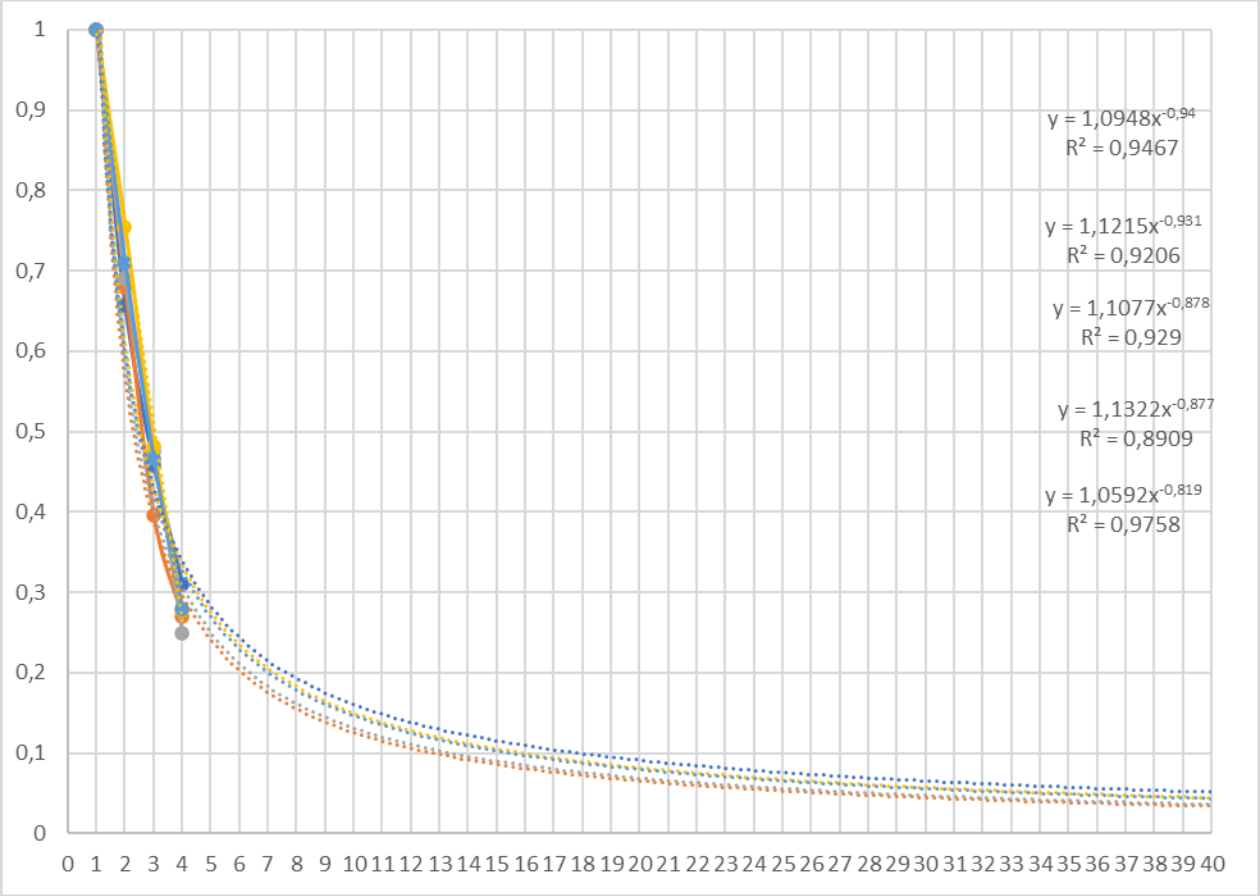
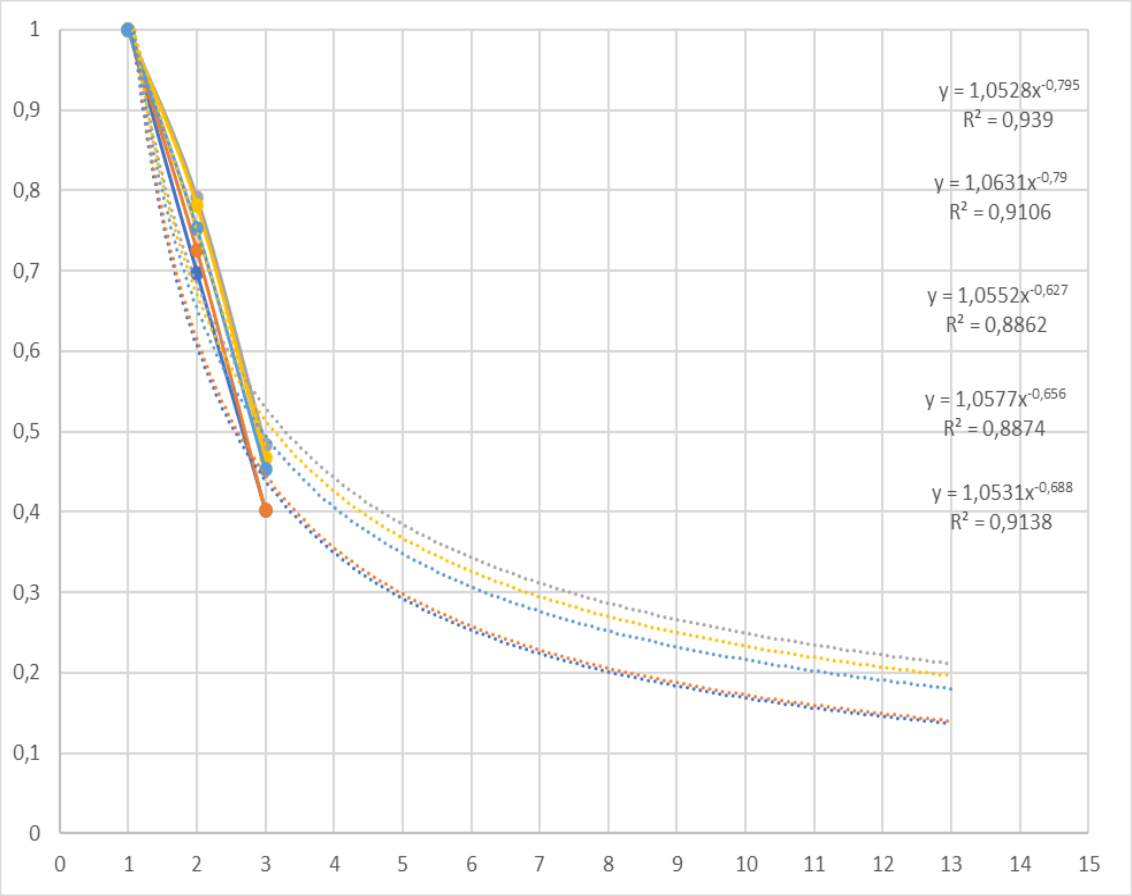
Трасса WinRAR os_mode в миллиардах инструкций



Пример расчета продуктивности

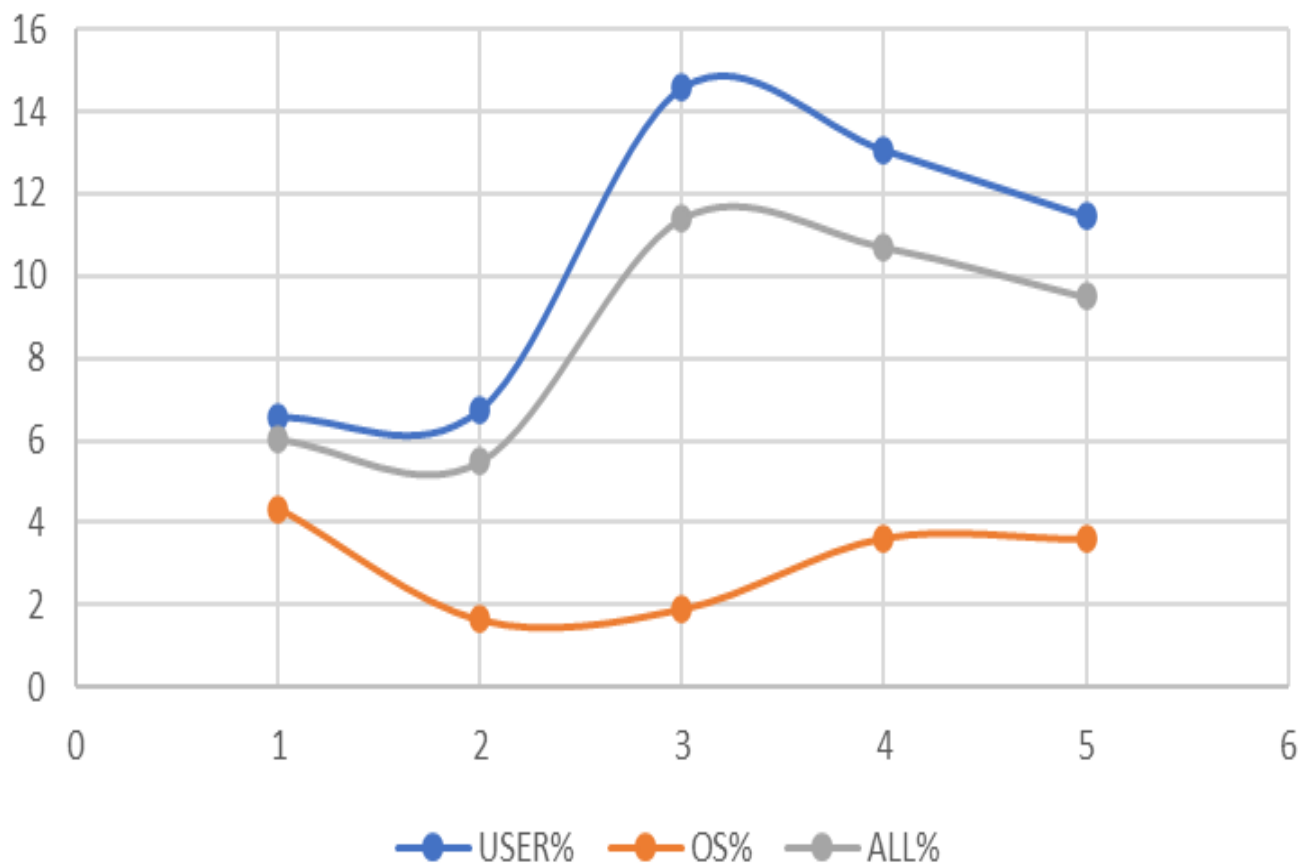
	us_mode				
Ранг трассы	Скорый	Быстрый	Обычный	Хороший	Максимальный
1	1	1	1	1	1
2	0,697499	0,725655	0,790422	0,781196	0,752326
3	0,402452	0,401985	0,483543	0,467232	0,452837

	os_mode				
Ранг трассы	Скорый	Быстрый	Обычный	Хороший	Максимальный
1	1	1	1	1	1
2	0,65645	0,679856	0,691044	0,754364	0,709582
3	0,458493	0,396083	0,475166	0,482544	0,466519
4	0,309919	0,269384	0,249902	0,277639	0,279319



Расчетная продуктивность (мой прогноз по продуктивности user_mode=10% и os_mode=2,5% практически совпали с фактической оценкой!(t_us% и t_os% веса метрик по времени)

Исходная продуктивность программного кода WinRar (OS-mode и USER-mode)



Тип	USER%	OS%	ALL%	t_us%	t_os%
скор	6,58	4,35	6,0225	0,75	0,25
быстр	6,77	1,65	5,49	0,75	0,25
обычн	14,58	1,91	11,4125	0,75	0,25
хорош	13,07	3,63	10,71	0,75	0,25
макс	11,46	3,6	9,495	0,75	0,25
Среднее	10,492	3,028	8,626		

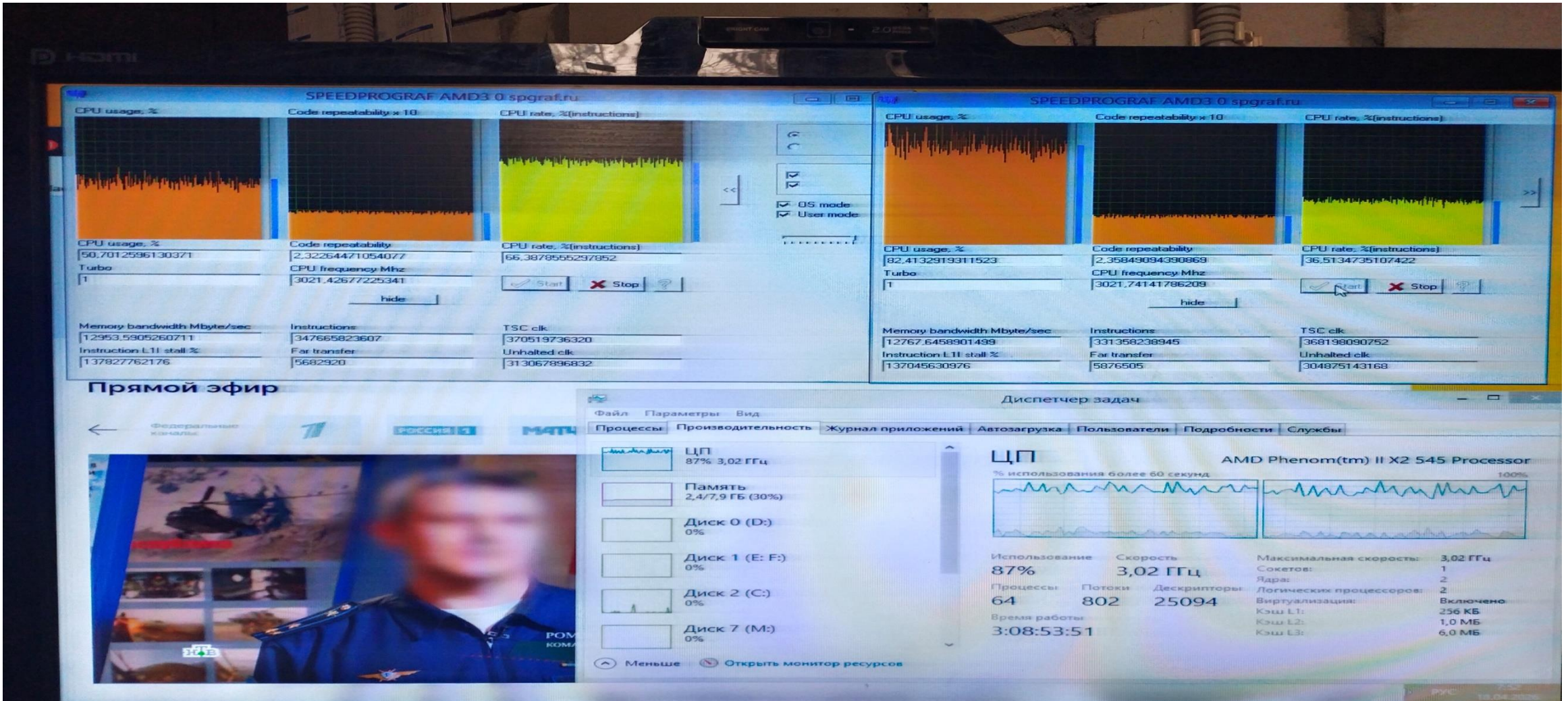
Структурные конфликты

- Я давно работал со своим профилировщиком, с разными платформами и процессорами и всегда оставалась какая-то неопределенность, не сходился баланс, пока я не добрался до структурных конфликтов (23-26 слайды с профилировщиками, без учета структурных конфликтов, там загрузка доходит до 100 %, но это обман!). Меня всегда смущал факт, почему 4 полных (без HT) процессора Intel при параллельной нагрузке 4-мя архиваторами загружены на 50% в режиме ядра? Сейчас об этом!
- 3 главных структурных конфликта (когда встает конвейер любого процессора):
 - Обращение за очередной командой, которой нет в кэш инструкций (Instruction fetch stall – IFS) самая массовая (80%);
 - Конфликт по управлению (неверное предсказание перехода) (10 %);
 - Конфликт по данным (требуется чтение результата предыдущей команды, но еще не произошла его запись в регистр буфера переупорядочивания, 1 такт надо еще подождать) (10%).

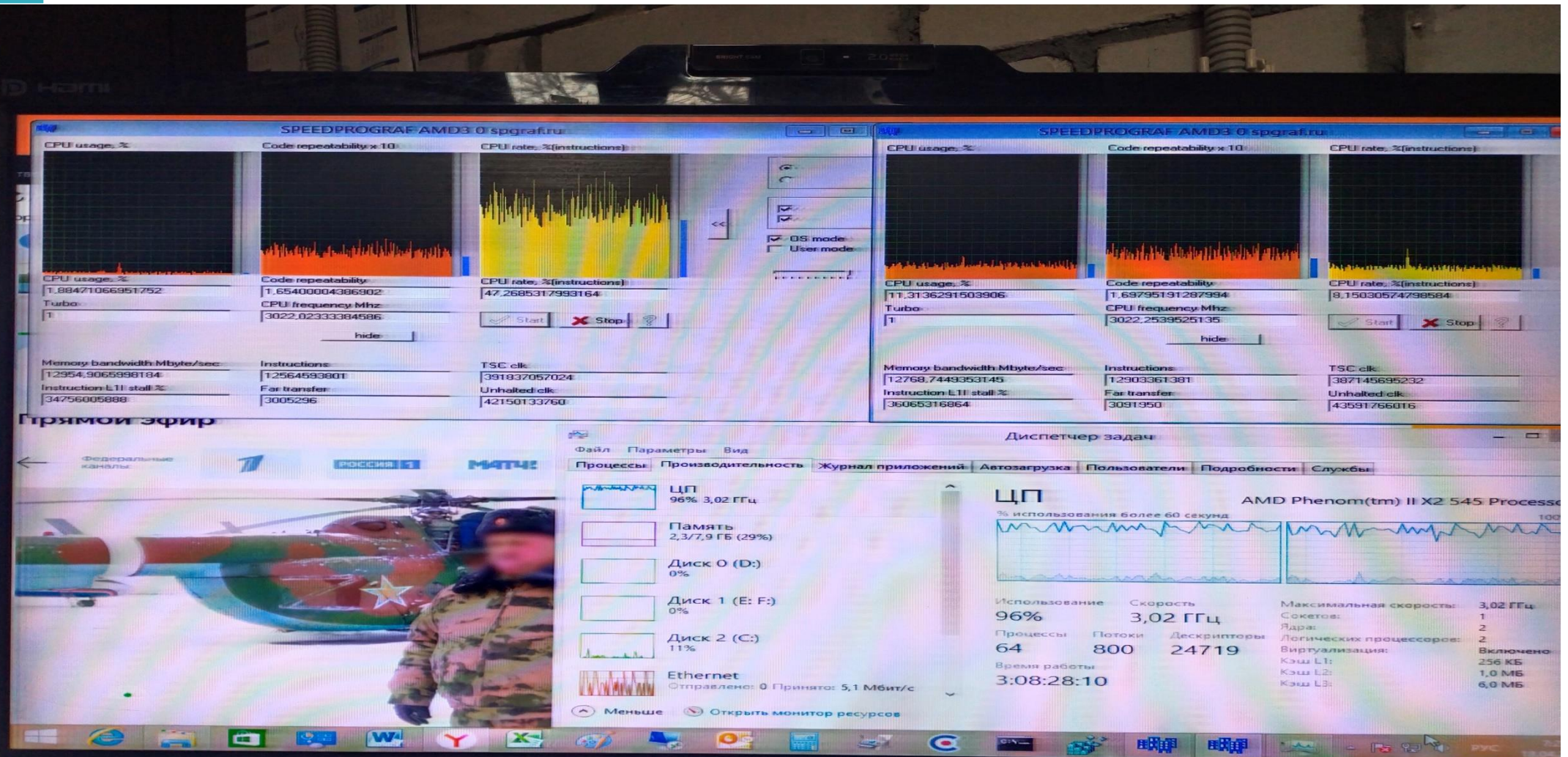
Первый конфликт полностью устраняет моя технология (IFS) – введением повторяемости кода блочной обработкой! Вторые два конфликта при потоковых вычислениях и обращения к базам данных составляют незначительную величину.

Я столкнулся с темой намеренного замалчивания этой темы! Во первых, свой профилировщик я для этого и написал! Профилировщики от производителей процессоров AMD и Intel (AMD Catalyst и Intel Vtune Amplifier(Analyzer)) не дают возможности профилировать ядро (ядро – священная корова, профилируется только приложение) и откровенно дурят со скоростью исполнения кода. Speedprograf мне помог установить реальную скорость исполнения и длину исполняемого кода в режиме ядра и в пользовательском режиме отдельно и вместе. Начиная с процессоров Nehalem Intel ввел «архитектурный мониторинг», в котором добавил пару MSR (наконец-то) регистров и несколько версий для регистрации событий гипертредовых процессоров и событий «не ядра». Но в базовых метриках архитектурного мониторинга (их 6) не осталось метрик конфликтов. Т.е.с помощью «архитектурного мониторинга» вы структурные конфликты не достанете в новых процессорах. А в гипертрейдинговых процессорах возникают еще и конфликты между тредами, они используют одни и те же ресурсы (**еще один 4-ый структурный конфликт между тредами!**). И поэтому достоверно выяснить, что происходит в многоядерном процессоре с включенным гипертрейдингом практически невозможно (много сопутствующего шума – структурных конфликтов). Даже в профилировщиках Linux_a есть строки по конфликтам Fron_end (выборка команд – про них Intel тихо молчит) и Back_end (исполнение – а про эти, девать некуда комментарии в Инете, хотя толку для разгона от них мало), но в современных процессорах может выйти надпись (not supported). Я просто сразу отключаю всякий гипертрединг и «неахитектурным мониторингом» достаю нужные метрики, в тех моделях, где они еще остались и доступны! Приступим.

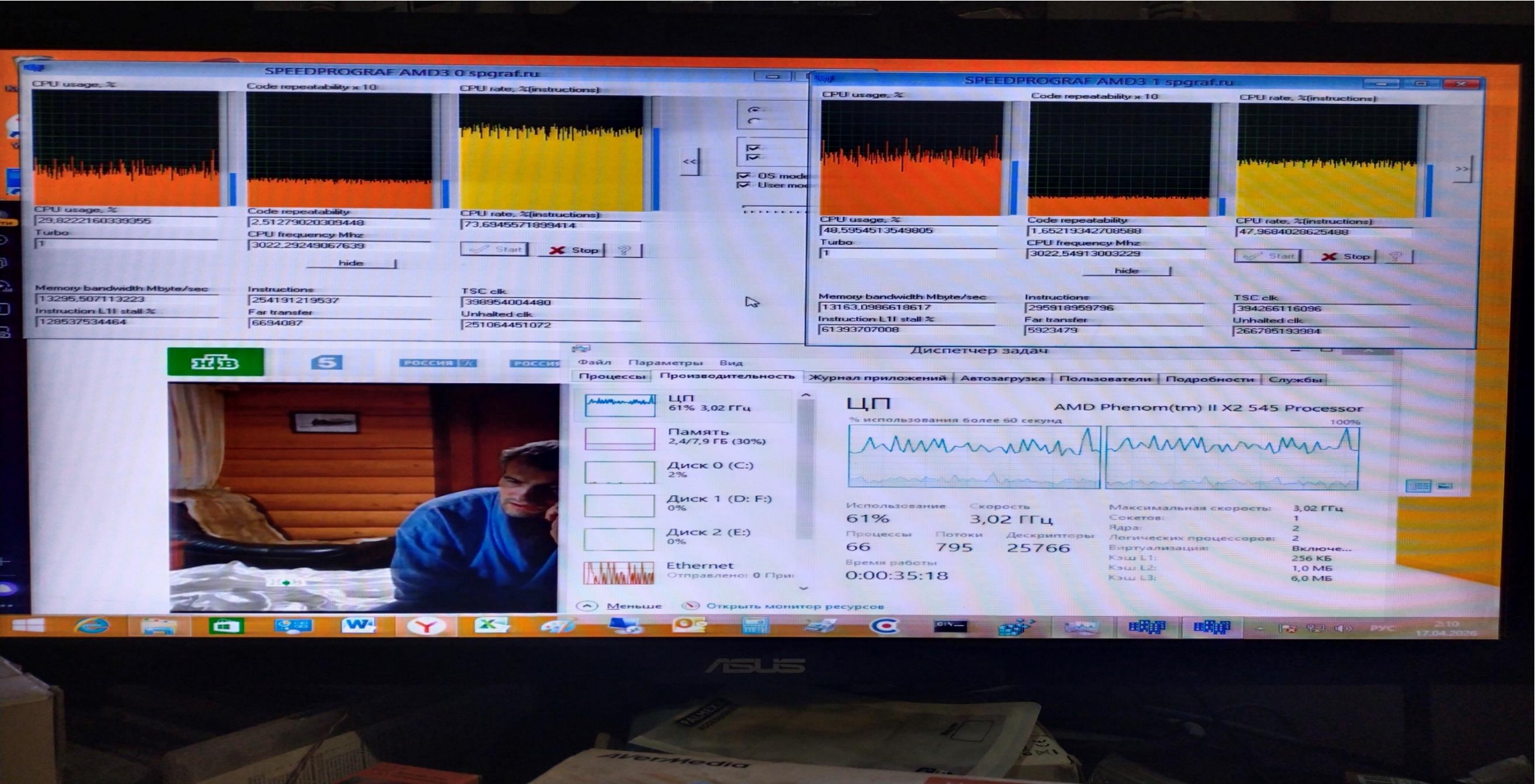
Вот пример получения трансляции через web-browser на двухъядерном процессоре AMD. Правый профилировщик Speedprograf выдает загрузку (CPU Usage%=82%) и считает быстродействие, как это делает «Диспетчер задач» (ЦП=87%), а левый Speedprograf вычитает структурные конфликты IFS (CPU Usage%=50%). На «Диспетчере задач» загрузка в режиме ядра и суммарная по ядрам одинакова. CPU Rate%=36% и CPU Rate%=66% слева и справа соответственно (процент пикового быстродействия) есть разница? Кто-то может сказать: «Да ладно, в пару раз, стоит ли ломать копыта?». Но если мы поделим Instructions L1 Stall/Unhalted clk ($137827762176/313067896832=0,44$), то получим 44% времени простоя конвейера процессора (у правого 44,9%), причем оба стояли практически одинаково! Половину времени конвейеры обоих ядер процессоров стояли!! Идем дальше.



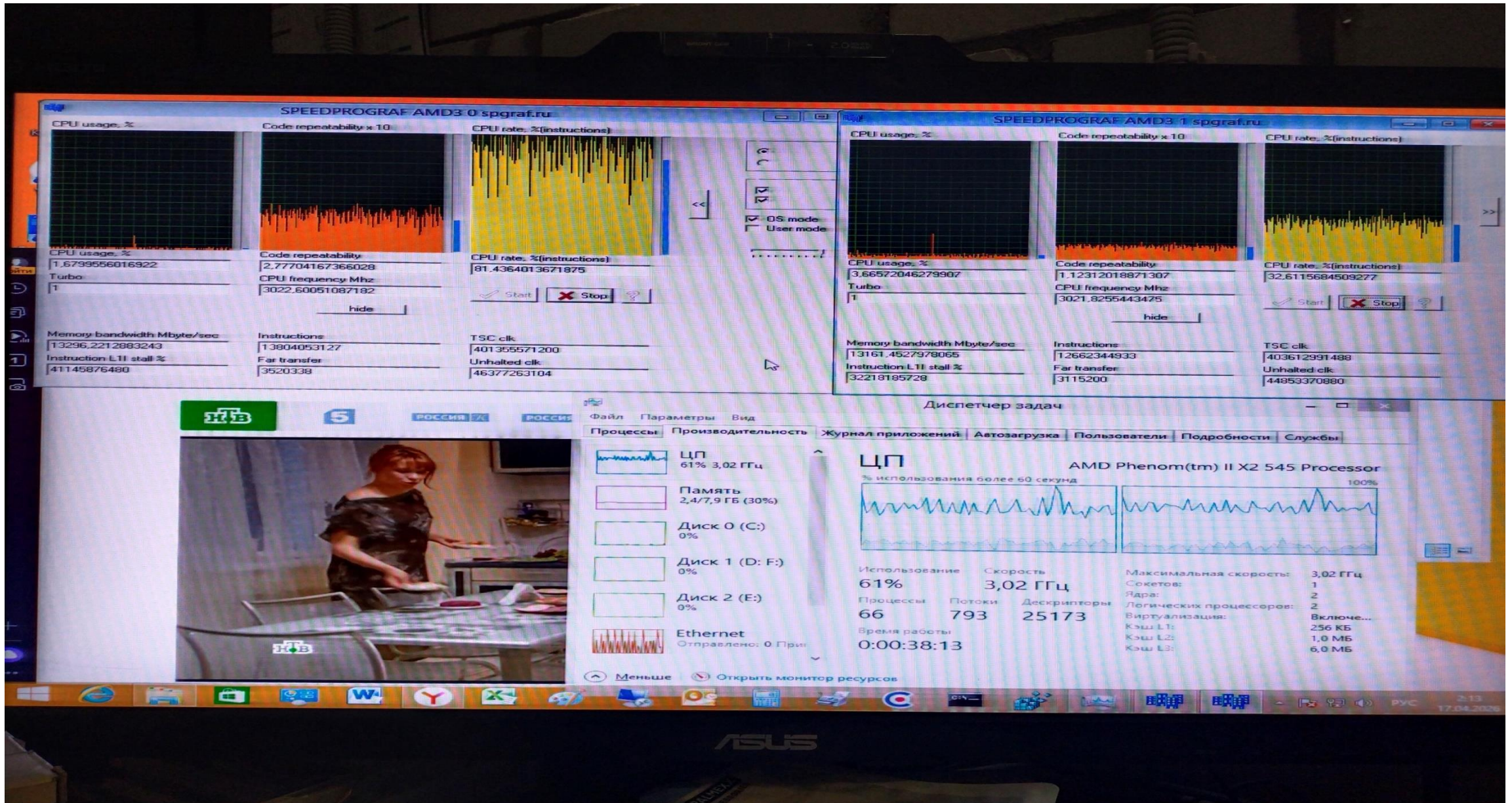
А это работа процессоров в режиме ядра (os_mode). Левый профилировщик Speedprograf выдает загрузку (CPU Usage%=1,88%), а правый Speedprograf выдает (CPU Usage%=11,31%). CPU Rate%=47% и CPU Rate%=8,1% слева и справа соответственно (процент пикового быстродействия) есть, разница в 5,8 раз? Вот тут можно начинать снимать «лапшу с ушей»! Если мы поделим Instructions L1 Stall/Unhalted clk (34756005888/42150133760=0,82), то получим 82% времени простоя конвейера процессора(у правого 83%), причем оба стояли практически одинаково! Больше 80% времени конвейеры обоих ядер процессоров стояли!! В чем тут истина, в том, что процессор в режиме ядра работает с 55% пикового быстродействия! Респект и уважение инженерам, создавшим процессор! А программистам, написавшим такое ядро, хвалиться нечем!! Хочу обратить внимание на одну деталь – время работы одного процессора в режиме ядра составило (CPU usage%) 11,3%. Если 100/11,3=8,8. Лет 15 назад у Intel возникала мысль ограничить количество ядер на 1 сокет 8-ю, поскольку прерывания на сокет обрабатывает только 1 процессор, а остальные при этом стоят.



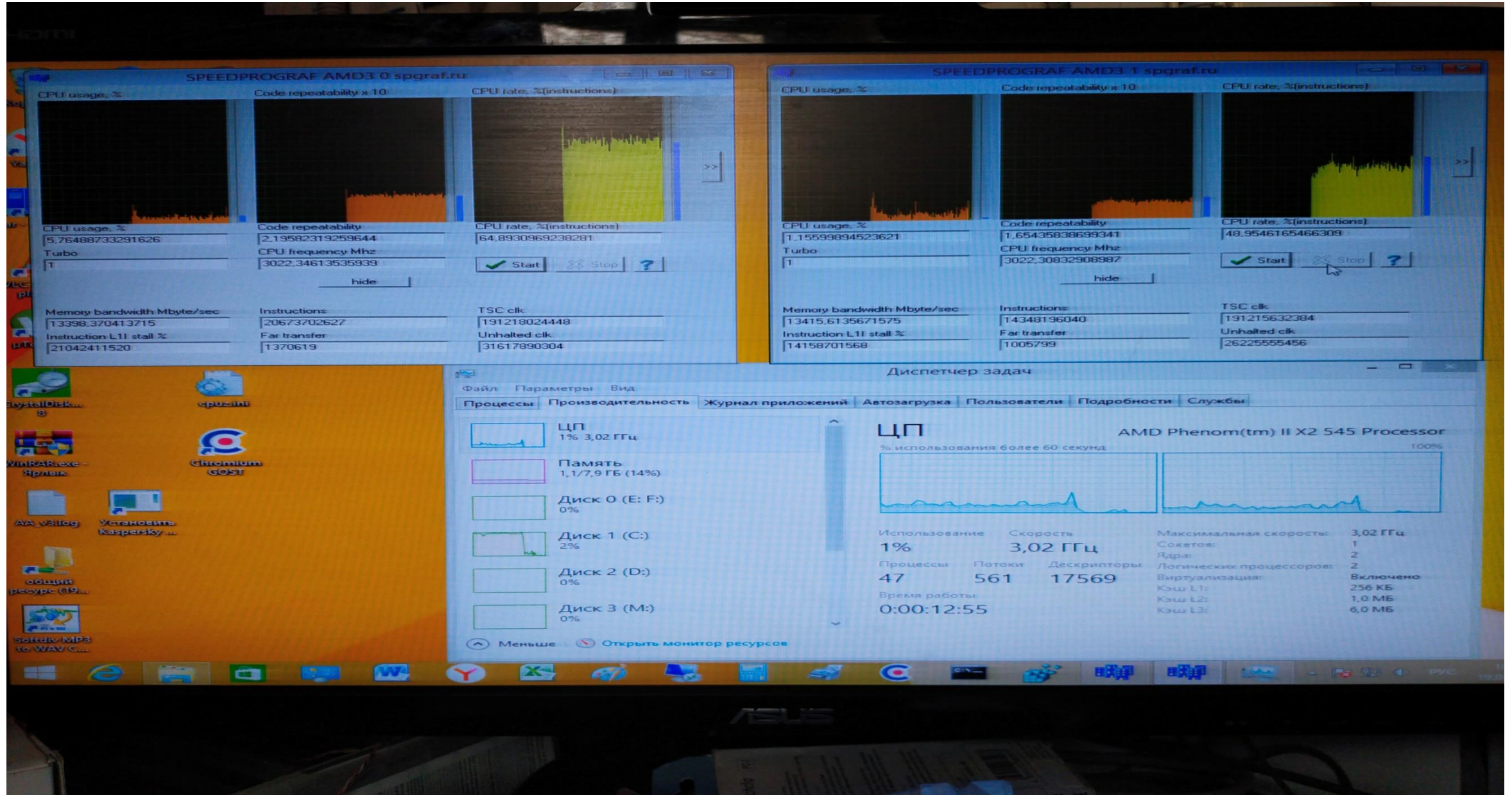
Теперь мы запустимся с SSD SATA диска, который в 3 раза быстрее. Оба профилировщика Speedprograf учитывают структурные конфликты IFS. По «Диспетчеру задач» видно, что общая загрузка снизилась с 87% до 65%, а загрузка ядра выросла в пару раз. На SSD диске разные ядра работают с различной скоростью. Средний график Code repeatability (повторяемость код в разах) показывает, во сколько раз код выполняется быстрее установленной памяти (тестирование которой производится при запуске профилировщика) или сколько он раз повторяется из кэш-памяти. В предыдущем случае значения были 2,32–2,36, сейчас 1,65–2,51? Более медленный процессор исполняет длиннее код (окно Instructions) 295518959796/254191219537=1,16. Простой конвейера составили у быстрого 51%, у медленного 23%.



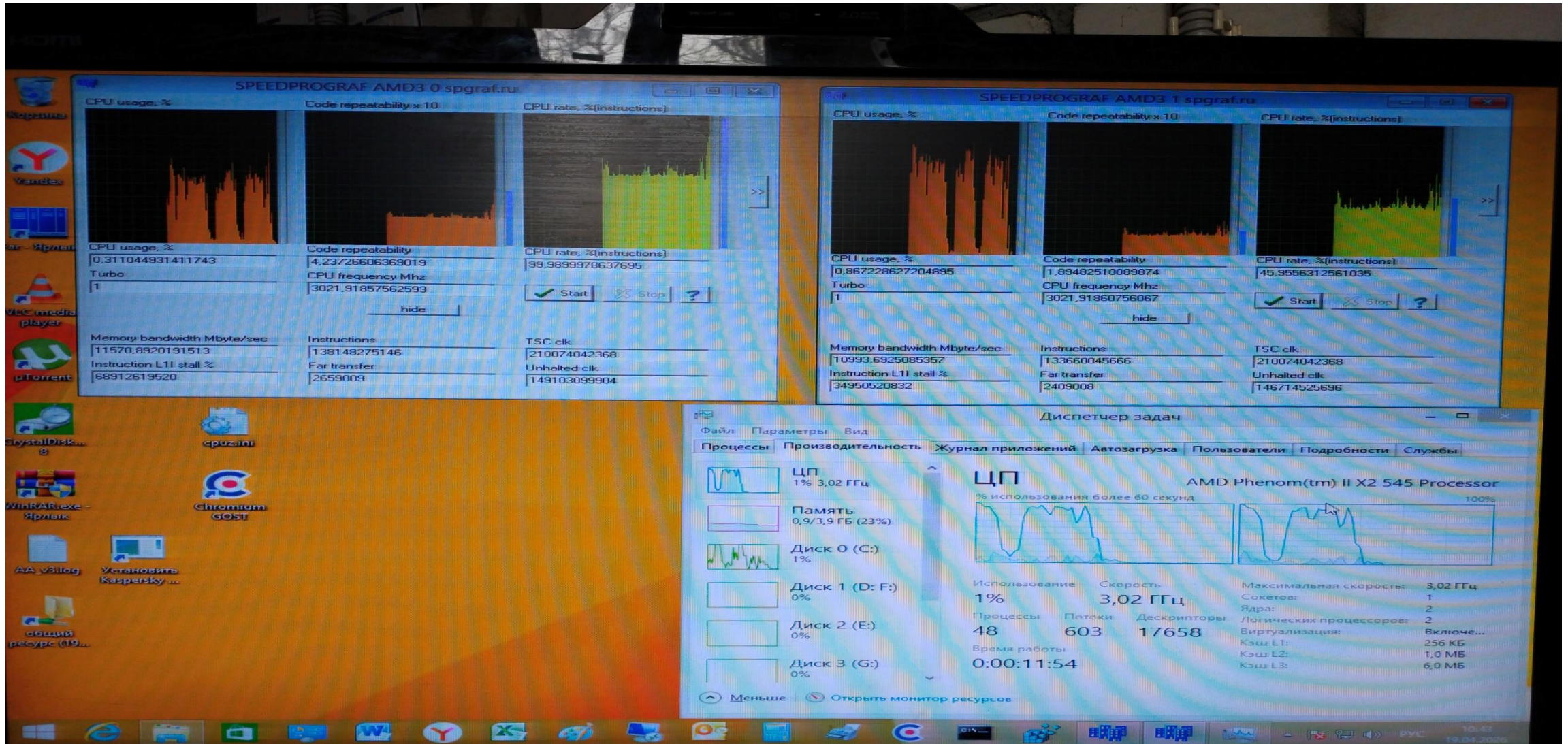
А это режим ядра на SSD. На быстром ядре код выполняется практически с пиковым быстродействием(>81%), при повторяемости 2,77, а на медленном скорость 32,6% при повторяемости 1,12. Длина кода у быстрого оказалась больше, 13,8 млрд. инструкций, против 12,7 у медленного. Структурные конфликты составили соответственно 88,6% и 71,8%.



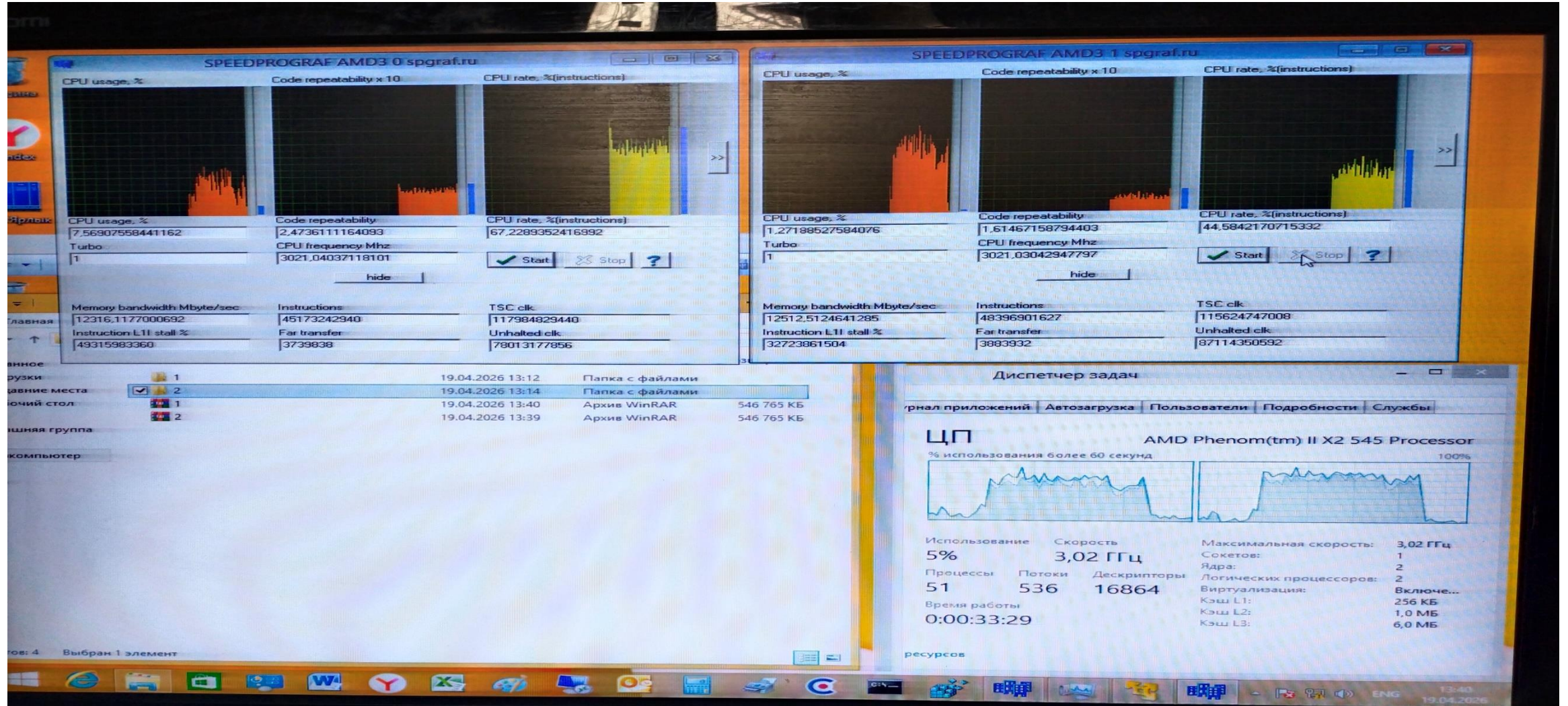
Главная функция операционной системы передавать данные приложению от устройств и носителей и забирать полученные результаты обратно для передачи в сеть или на внешние носители. Ядро операционной системы работает не только в режиме ядра, но и в пользовательском режиме, поскольку все модули (*.dll файлы) существуют в 2-х вариантах – для режима ядра и для пользовательского режима. Здесь приведены данные по переносу данных приложению (534Мбайт) с внешнего носителя и запись обратно на диск в другой файл (архивация без сжатия) с ATA диска двумя ядрами, у обоих одинаковые задания. Время переноса составило: (млрд. TSC/3) $191,2/3=63,7$ сек. Загрузка с простоями ядер составила $\text{Unhalted clk/TSC clk}$ $31,6/191,2=16,5\%$ и $26,2/191,2=13,7\%$, в среднем $15,1\%$ на ядро. Суммарная трасса двух ядер составила 35 млрд. инструкций. Ограничивающим ресурсом оказался ATA диск, процессоры простаивали 85% времени в ожидании его реакции.



Такое же задание на SSD SATA (в 3 раза быстрее ATA диска, 450 Мбайт/с против 160 Мбайт/с). Время 70 сек (даже медленнее на 7 сек), загрузка с простоя средняя 70,4% (в 4,66 раза больше?), суммарная трасса – 133,66+138,14= 271,8 млрд. инструкций (в 7,8 раз больше?). Тут есть вообще вопросы по замиранию вычислений в ноль, но сейчас не об этом. SSD диск загружает ядра больше. Но они и команд выполнили в 8 раз больше?!? Это как? За что? А сделали то же самое, да еще и медленнее! Простой переход на SSD диск не ускорил работу с данными, а даже замедлил и усугубил увеличением загрузки ядер!



А здесь то же задание в виртуальной среде на SSD ATA диске. Время среднее 38,9 сек (в 1,64 раза быстрее варианта ATA диска), загрузка с простыми средняя 70,7% (такая же, как в предыдущем варианте), суммарная трасса – 93,6 млрд. инструкций (в 2,7 раз больше первого и в 2,9 раза меньше второго варианта). «Виртуальная машина» слегка притормозила SSD диск и сократила трассу в 3 раза, а время исполнения почти в 1,6 раза. Предлагаемая мною технология как раз и сокращает длину кода до минимума, но не благодаря тормозам производительности, а как встроенный инструмент контроля накладных расходов в ядре операционной системы, сокращающий только длину исполняемого кода в 5–29 раз! Диски SSD.M2(Nvme) еще больше усугубляют обстановку по накладным расходам! На слайде 23 SSD.M2(Nvme) на памяти DDR4, но структурные конфликты уперлись в «стену памяти». Память держит!

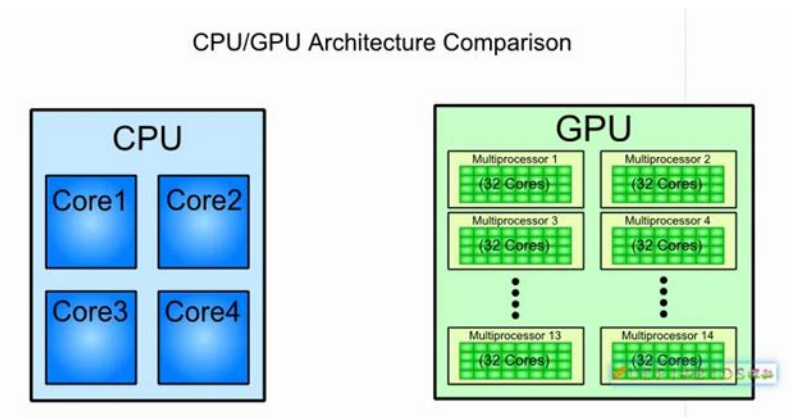
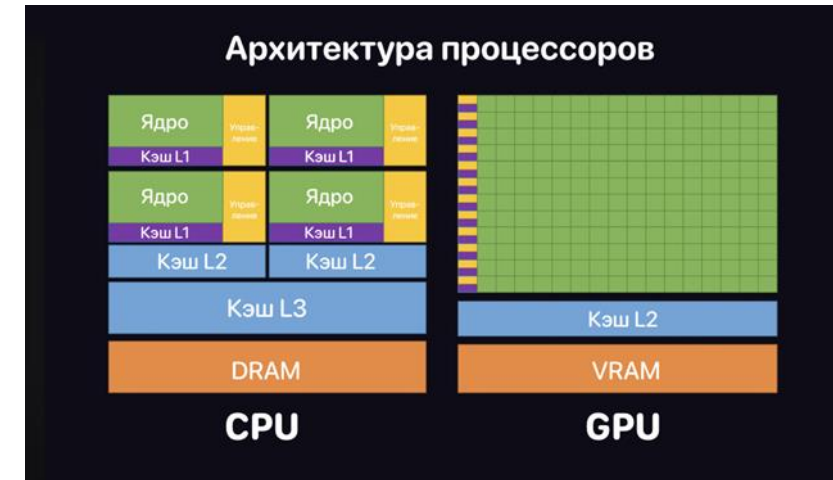


Speedprograf, итоги и предложения

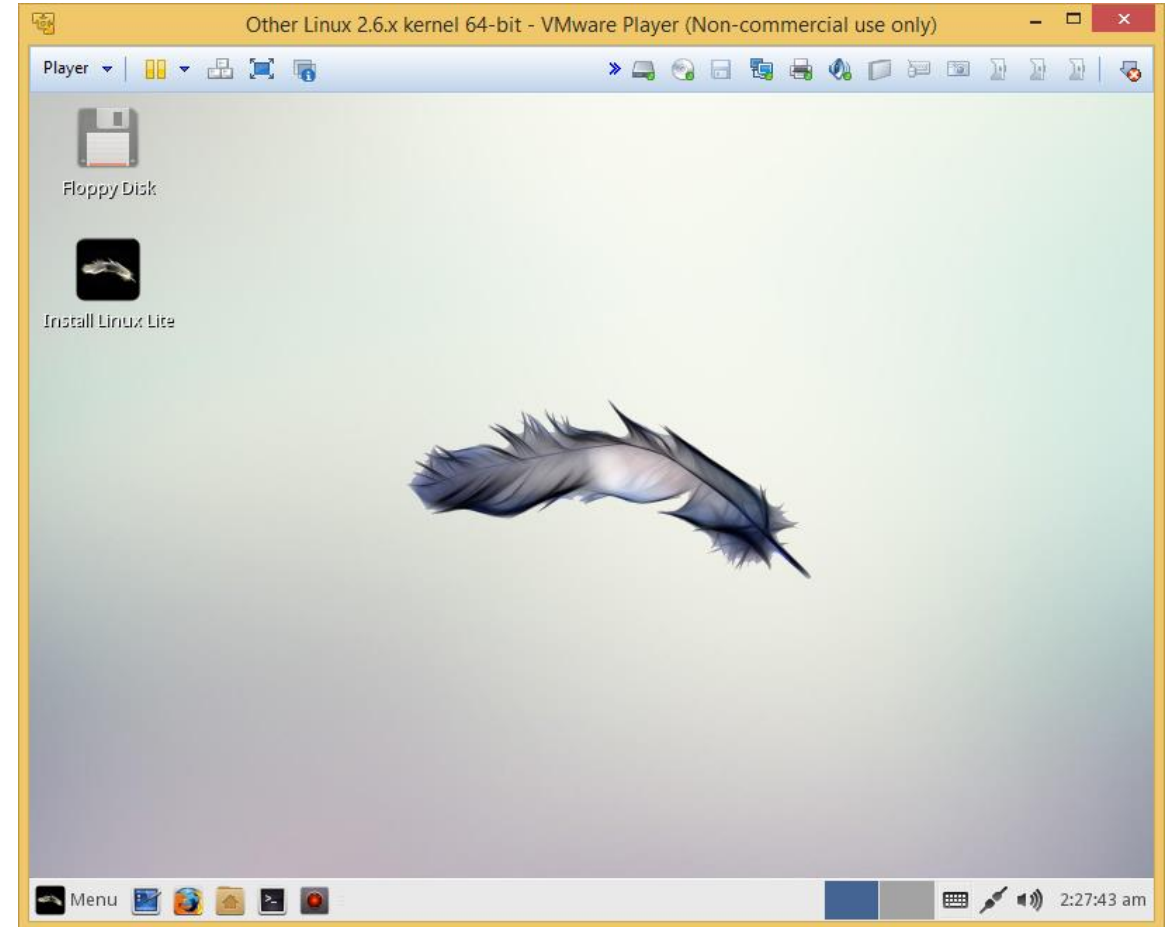
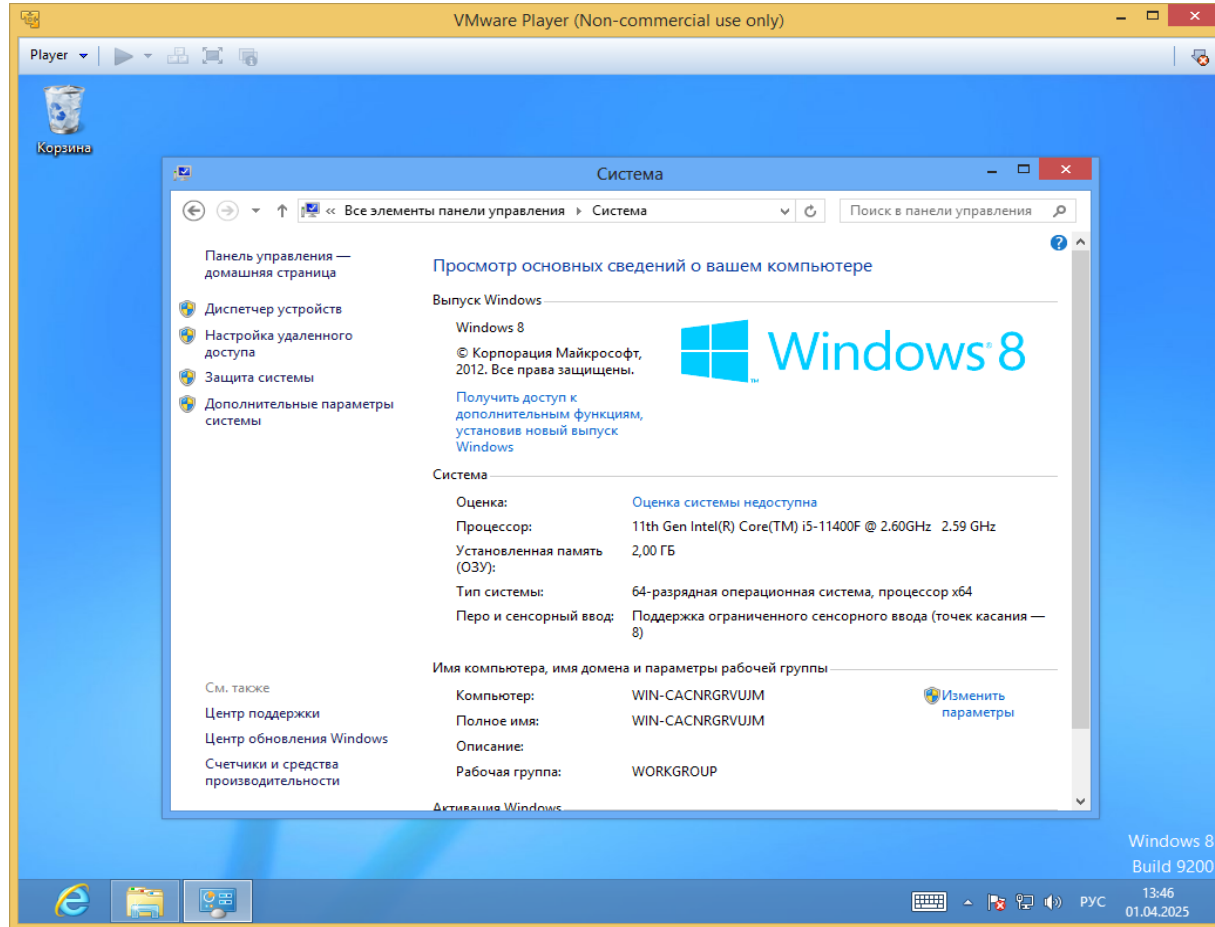
- Мой профилировщик Speedprograf (Зарегистрирован в Реестре Российского ПО под номером 22515 – <https://reestr.digital.gov.ru/search/?q=22515>) подтверждает наличие проблемы «стены памяти» в сегменте ПК и ПО общего назначения.. Бесплатный Speedprograf для процессоров Intel Core IX для 8 ядер, Intel Pentium IV для 2-х ядер и для двухядерного AMDK10 выложен на моем сайте spgraf.ru. Трасса 1-го порядка пользовательского режима (однопроходной код) всегда выполняется со скоростью памяти и практически без структурных конфликтов конвейера ядра. Трассы 2-го, 3-го и 4-го порядков могут выполняются со скоростью в 2-3 раза превышающей скорость памяти, вплоть до пикового быстродействия (без учета циклов простоя конвейера), но с большим уровнем структурных конфликтов. Допустим код, исполняющийся в 2,5 раза быстрее скорости памяти, будет иметь половину таков простоя и в результате средняя скорость исполнения составит 1,25 скорости памяти. Суммарный код (пользовательский режим и режим ядра) со структурными конфликтами по средней скорости приближается к скорости памяти. Проблема процессора не в реализации пикового быстродействия, когда конвейер не стоит, он легко работает на пиковом быстродействии даже в режиме ядра. Проблема в структурных конфликтах при обращении к памяти за исполняемым кодом. Если код сделать повторяемым, структурные конфликты исчезнут и сбудется места академика Бурцева Всеволода Сергеевича об автоматической синхронизации кода и данных. Программист не в состоянии этим заниматься, эта принудительная процедура должна быть реализована в ядре. Ваше приложение может оказаться в разных условиях загрузки и в окружении других приложений, предвидеть заранее возможные комбинации которых просто невозможно. Процедура контроля принудительного повторения кода в ядре должна быть автоматической. Код ядра исполняется со скоростью 0,2-0,4 скорости памяти, но уровень структурных конфликтов может достигать (простоя) 80-95%. Я могу допустить, что на DDR2 памяти 2 ядра будут простаивать половину времени, на DDR3 4 ядра будут простаивать столько-же, на DDR4 – 8 ядер, на DDR5 –16. Но вот что будут делать на DDR5 24 ядра и 48 потоков я представляю себе с трудом, особенно как будет выглядеть синхронизация обращения к общим ресурсам? Я не привел данные по виртуальным машинам, но там хост плейера одного ядра забирает 30-40% времени в режиме ядра. Перегрузка ядер кодом ядра чревата параличом вычислений (для гипервизоров это актуально).
- Предлагаемый способ ускорения путем повышения продуктивности программного кода имеет потенциал от 45 раз (user_mode) до 130 раз (os_mode) . Мой подход дает результат выше, чем прогноз по увеличению реализации быстродействия суперкомпьютеров в 10-40 раз(американская оценка). Решение проблемы накладных расходов имеет значимость на порядок превышающую реализацию преимуществ иерархии памяти процессоров (кэш дает выигрыш всего 2-3 раза). Приведенные в презентации изменения длины исполняемого кода – это ограниченная реакция современных ОС не благодаря их положительным свойствам, а вопреки – это реакция на замедление работы платформ в ограниченных пределах. Накладные расходы нивелируют производительность информационных систем – медленные системы частично исполняют код короче, высокопроизводительные системы – больше производят непроизводительных действий! Если взять иерархию аппаратных платформ по скорости работы памяти архитектуры X86_64 DDR2, DDR3, DDR4, DDR5, то ситуация выглядит следующим образом: варианты комплектации дисковыми подсистемами ATA дисками, SSD ATA, SSD M2, RAM-дисками с памятью DDR2 обнаруживают трассы от 1 до 4 порядка, с DDR3 – от 1 до 3 порядка, с DDR4 – от 1 до 2 порядка, с DDR5 – только 1 порядок! Трасса первого порядка самая длинная и обладает исходной продуктивностью 10%. Эволюция SSD носителей и оперативной памяти DDR2->DDR3->DDR4->DDR5, а также увеличение кэша современных процессоров от 100 (Intel) до 400 (AMD) Мбайт в бытовых ПК привела к ускорению «начальной загрузки» и к восстановлению накладных расходов «первого прохода»! Мы ускорили начальную загрузку и первый проход, но не работу вычислительной системы. Низкопродуктивный код никуда не делся! Без решения проблемы накладных расходов вы обрекаете современные вычислительные ресурсы на работу в 10-100 раз медленнее, несмотря ни на какие современные аппаратные ускорения платформ! Без предлагаемой мною блочной обработки эта проблема не может быть решена в принципе и альтернативного пути просто нет!
- Для разработки макета ядра с повышенными потребительскими свойствами ищу партнера с возможностью консультаций на уровне разработчика по своему ядру! Технология блочной обработки применима к любому процессору, независимо от места его применения и масштаба задач, от ПК, серверов , гаджетов, приборов, беспилотных транспортных средств и дронов до ЦОДов и суперкомпьютеров.

Замечание по суперкомпьютерам

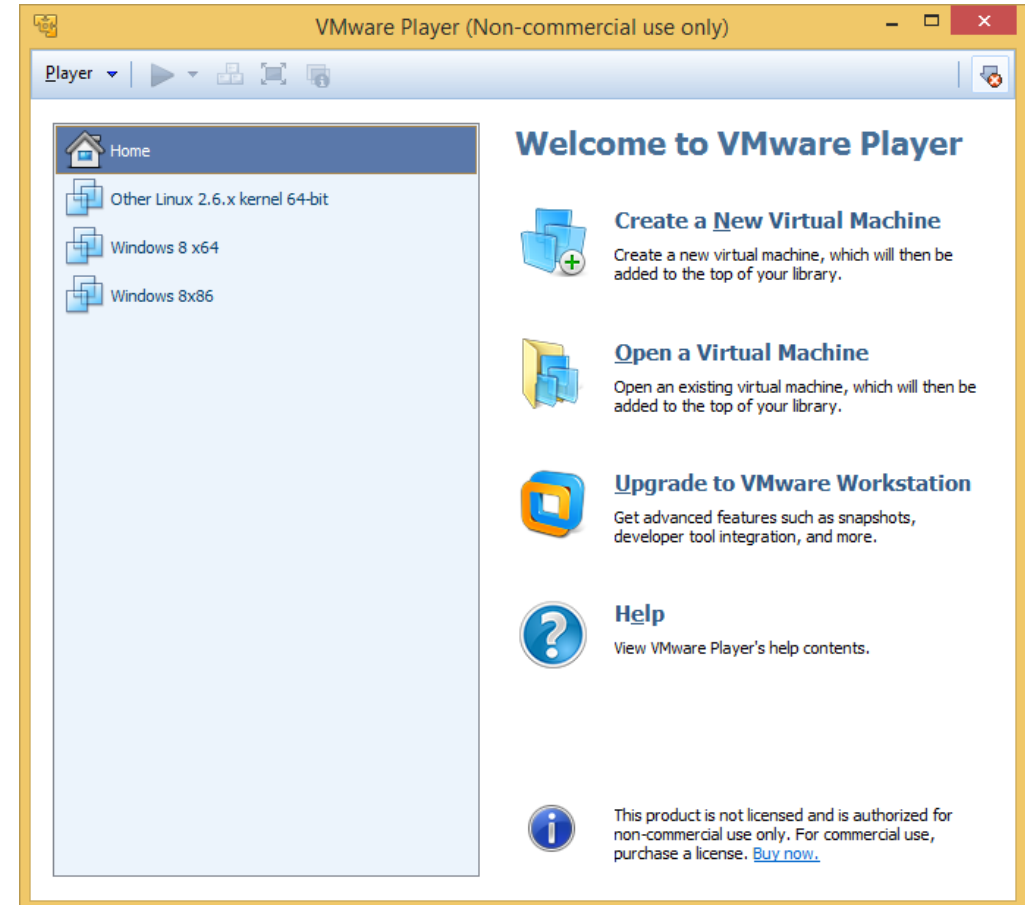
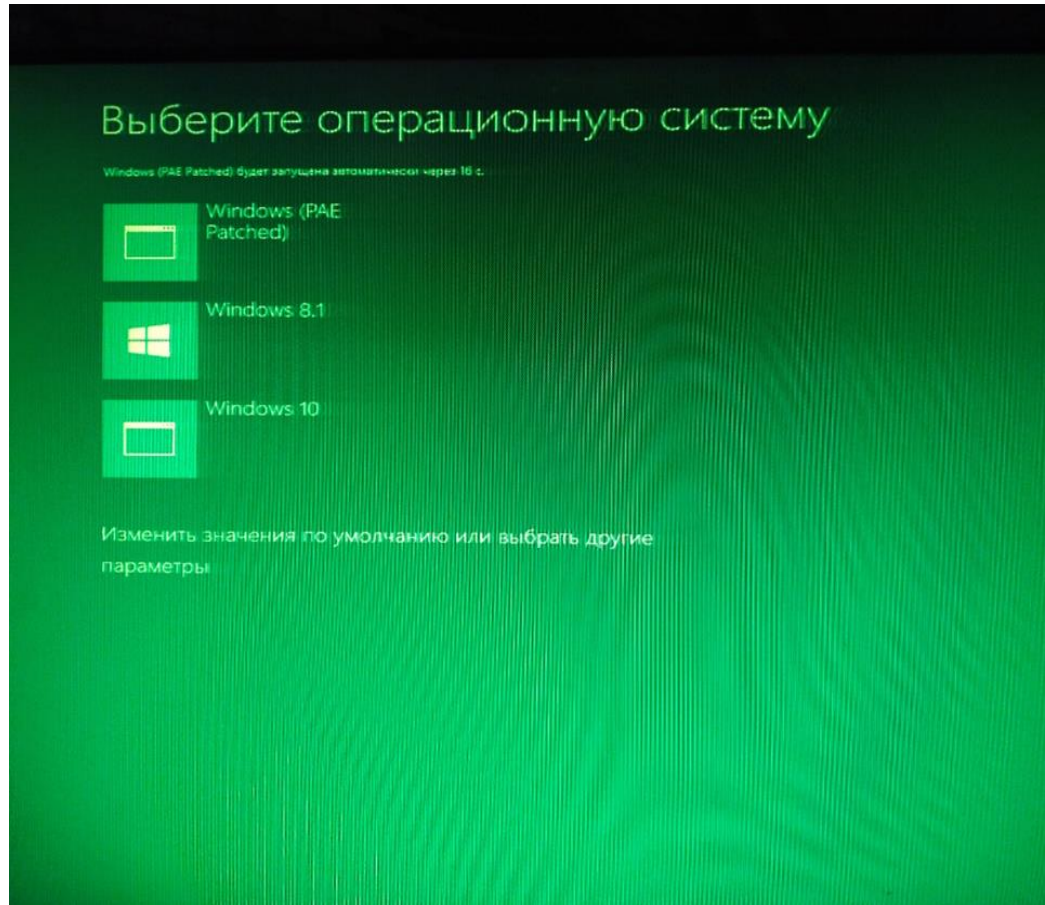
- В США существовала программа развития суперкомпьютеров DARPA HPCS (2001-2010 годы), в которой ставилась задача увеличения производительности суперкомпьютеров на реальных приложениях в 10-40 раз (не за счет увеличения количества ядер, а за счет повышения реализации их пикового быстродействия). Задача увеличения реальной производительности не была выполнена, поскольку уперлась в «стену памяти» и перешла для решения в очередную программу США по суперкомпьютерам UNHPCS (2011-2020 годы). На сегодняшний день эта задача так и не решена. Но если в 10-40 раз можно увеличить реальную производительность для суперкомпьютеров, то их реальный текущий КПД по быстродействию 2,5-10%. Список ТОП-500 суперкомпьютеров оказался не у дел при майнинге криптовалют – затраты на электричество превысили объем сгенерированной криптовалюты. Для генерации криптовалюты на смену им пришли ASIC-майнеры, которые больше ни для чего не пригодны!
- Предлагаемый мною способ блочной обработки (с кратностью от 10 до 40) может быть применен с еще большей кратностью для новых широкополосных видов памяти HBM для графических процессоров (GPU) и суперкомпьютеров. HBM4 (High-Bandwidth Memory 4) — это стандарт памяти четвертого поколения из семейства HBM для GPU (ширина 2048 бит). Некоторые особенности памяти HBM4: высокая пропускная способность — в отличие от DDR5, которая может обеспечивать скорость до 51,2 ГБ/с на модуль, HBM4 предлагает пропускную способность, превышающую 1 ТБ/с на стек (а стеков 16). Разрядность шейдерных и полигональных процессоров видеокарт (GPU) 8-16-32 бита?!?. Новые чипы NVIDIA и AMD, будет производить Тайваньский интегратор TSMC по их дизайну и монтировать память Samsung и Hynix на них. Без применения предлагаемого способа ускорения тоже работать эффективно не будут, поскольку дисбаланс ширины шины памяти и разрядности процессоров достигает, как минимум 64-х раз! Очевидно без многофазной синхронной или асинхронной блочной обработки многослойных моделей ИИ здесь не обойтись и видимо будут разработаны специальные численные методы решения задач ИИ блоками заданной размерности для GPU и суперкомпьютеров. В противном случае, шина памяти с большой шириной будет просто простаивать или работать неэффективно!



Профилировать можно гостевые операционные системы любой разрядности (и 32 и 64 бита) и любых разработчиков, которых поддерживает система виртуализации. В качестве примера приведены 64-разрядные



Для использования Speedprograf_a на платформе 32-бит используется PAE (Physical Address Extension) технология для подключения современных платформ с необходимым объемом памяти. Для профилирования гостевых операционных систем разрядностью 32 и 64 бита необходимо на хостовой машине запустить поддерживаемую на хосте систему виртуализации (аппаратная платформа при этом должна поддерживать технологию VT-X (Intel VT или AMD-V)), установить гостевую операционную систему по вашему выбору.



Запуск на любой платформе профайлера Speedprograf возможен с поддержкой BIOS\legacy с SSD.M2 Nvme.



РОССИЙСКАЯ ФЕДЕРАЦИЯ



СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022682097

Монитор реального быстродействия и профилировщик
процессоров серии X86_64 Speedprograf

Правообладатель: *Суворов Дмитрий Анатольевич (RU)*

Автор(ы): *Суворов Дмитрий Анатольевич (RU)*



Заявка № 2022669865

Дата поступления 25 октября 2022 г.

Дата государственной регистрации

в Реестре программ для ЭВМ 18 ноября 2022 г.

Руководитель Федеральной службы
по интеллектуальной собственности

Документ подписан электронной подписью
Сертификат 68b80077614c1932094e8b24145d5c7
Владимир Зубов Ю.С. Зубов
Действителен с 25.10.2022 по 25.05.2025

Ю.С. Зубов

SPEEDPROGRAF



Спасибо за внимание!