

Отчет о содержании документа «CIPR 2026_SuvorovDA.pdf»

Название: Повышение производительности прикладного ПО общего назначения процессоров x86_64 серии путем доработки системного ПО. Универсальный способ ускорения вычислений на этапе исполнения.

Автор: Судя по тексту, документ подготовлен Д.А. Суворовым (SuvorovDA).

Основная идея документа: Автор предлагает и обосновывает универсальный способ значительного (на 1-2 порядка) ускорения вычислений на процессорах архитектуры x86_64. Способ заключается в снижении накладных расходов за счет применения **пакетной (блочной) обработки данных** на этапе исполнения, изменяя организацию вычислений без модификации функционального программного обеспечения.

Структура и ключевые тезисы отчета

1. Проблема: Критически низкая продуктивность программного кода

- Проводится аналогия между КПД тепловых двигателей и продуктивностью ПО.
- Утверждается, что исходная продуктивность современного ПО (отношение полезных команд к общему числу команд) составляет всего **2.5–10%**, что хуже КПД паровоза (5–12%).
- Прослеживается эволюция падения продуктивности по поколениям ЭВМ: от 33% для ПО 2-го поколения (в машинных кодах) до 2.5% для ПО 4-го поколения (режим ядра, ООП). Основная причина — рост накладных расходов (до 97.5%).

2. Исторические предпосылки и обоснование метода

- В 1990-е годы автор, занимаясь оценкой производительности ВК АСУ ПВО, обнаружил, что производительность комплексов недостаточна, а менять аппаратуру и переписывать ПО нет времени.
- Была выдвинута и успешно применена гипотеза: **ускорить вычисления путем сокращения накладных расходов** за счет изменения организации вычислений (пакетной обработки). Утверждается, что так было доработано 4 промышленных ВК с повышением производительности в **2–4 раза**.
- Ключевое наблюдение: при однократном запуске целевой функции полезная работа составляет лишь 33% кода, а 67% — накладные расходы. Многократное выполнение целевой функции (пакетная обработка) после одного подключения увеличивает долю полезной работы.

3. Технические причины накладных расходов на x86_64

- Иерархия описания функционала программ в системах программирования (объектно - ориентированное программирование) требует при исполнении кода программы определить место расположения кода и данных, сначала прочесть 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 то же 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 взаимодействия ядра и приложения с точки зрения затрат ресурсов не существует.

- Большая вложенность вызовов (16-18 уровней) и затратные операции сохранения/восстановления регистров (PUSH/POP): до 34 команд на один вызов.
- Разрушительное действие инструкции `syscall`: при переходе в режим ядра процессор «зачищает» конвейер, таблицу переходов и другие структуры, что заставляет его заново «обучаться» и загружать данные при возврате.

4. Суть универсального способа ускорения: Блочная (пакетная) обработка

- Вместо обработки блоков данных по одному с полным циклом накладных расходов для каждого, предлагается за один «подход» обрабатывать пакет из k блоков.
- При этом накладные расходы платятся один раз за пакет, а полезный код выполняется k раз, что резко повышает общую продуктивность.
- Приводится математическая модель (плоская модель накладных расходов):
 - $a = c/(n+c)$ – исходная продуктивность кода
 - $a_k = ka/(1+a(k-1))$ – продуктивность при пакете размером k .
- Это ведет к сокращению «алгоритмической емкости» (длины исполняемого кода).
- Дополнительный выигрыш дает кэш-память: повторяющийся код исполняется из кэша, минуя задержки оперативной памяти.

5. Количественные оценки и потенциал ускорения

- Приводятся расчетные таблицы, показывающие рост продуктивности и сокращение нормированной длины кода в зависимости от размера пакета k (до 10, 20, 40 блоков).
- Для пользовательского режима (исходная продуктивность ~10%): потенциал ускорения **до 45 раз**.
- Для режима ядра (исходная продуктивность ~2.5%): потенциал ускорения **до 130 раз** (с учетом разгрузки шины памяти).
- Утверждается, что этот подход на порядок эффективнее, чем иерархия кэш-памяти, дающая выигрыш лишь в 2–3 раза.

6. Инструмент анализа: профилировщик Speedprograf

- Автор демонстрирует данные, полученные с помощью собственного профилировщика Speedprograf (зарегистрирован в Реестре российского ПО, №22515).
- **Ключевое открытие:** Профилировщик выявляет «структурные конфликты» (в основном, остановки конвейера при промахах мимо кэша инструкций — Instruction Fetch Stall), которые не показывают стандартные средства вроде Intel VTune или AMD Catalyst, особенно для режима кода ядра.
- Профилирование показывает, что:
 - «Загрузка ЦП» в «Диспетчере задач» обманчива. Например, при отображаемой загрузке 87% реально конвейер процессора может простаивать **до 50%** времени в пользовательском режиме и **до 80-95%** — в режиме ядра.
 - Проблема не в пиковом быстродействии процессора, а именно в структурных конфликтах при обращении к памяти за кодом.

- Более быстрые платформы (SSD, быстрая DDR-память) парадоксальным образом могут **увеличивать** объем выполняемых накладных действий (трассу команд) для одной и той же задачи, нивелируя выигрыш аппаратуры .
- Ускорение памяти от DDR2 к DDR5 ведет к восстановлению самой длинной и низкопродуктивной «трассы первого порядка».

7. Связь с контейнеризацией и мировыми тенденциями

- Автор упоминает технологию контейнеризации как смежный способ снижения накладных расходов и заявляет, что его метод может на порядок повысить ее эффективность на этапе исполнения.
- Проводится параллель с программой DARPA HPCS (2001–2010), ставившей задачу повысить реальную производительность суперкомпьютеров в 10–40 раз. Задача не была решена из-за «стены памяти». Автор утверждает, что его метод способен решить эту проблему.

8. Применимость и поиск партнеров

- Технология применима к любым процессорам (x86, GPU) и любым масштабам систем: от дронов и гаджетов до ЦОДов и суперкомпьютеров.
- Для GPU с широкой памятью HBM4 предлагается использовать многофазную блочную обработку.
- Автор ищет партнера с доступом к разработке ядра ОС (гипервизора, контейнерной ОС) для создания макета ядра с «повышенными потребительскими свойствами». Несмотря на промышленную реализацию, в настоящее время технология практически нигде широко не используется, поскольку имеет некоторые нюансы своего применения, которые будут предоставлены при заключении партнерского соглашения автором.

9. Заключение

Без внедрения методов принудительного повторения кода (блочной обработки) и контроля в ядро ОС, проблема накладных расходов неразрешима. Современные вычислительные ресурсы работают в 10–100 раз медленнее своих возможностей, несмотря на любое аппаратное ускорение.

Предлагаемый способ — единственный путь преодолеть «стену памяти» и кардинально повысить реальную производительность вычислительных систем.

Открытые публикации автора и презентация CIPR 2026_SuvorovDA на премию AWARD выложены на сайте spgraf.ru/Publications.

Закрытых публикаций более 30 (список для ознакомления может быть представлен).