УДК 519.685
DOI: 10.26102/2310-6018/2026.58.7.009

Фазовая оптимизация однопоточного выполнения независимых программных последовательностей

Е.А. Буевич

Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация

Резюме. В работе рассматривается один из аспектов совместного выполнения в одном потоке нескольких независимых друг от друга по данным программных последовательностей, что достигается чередованием их инструкций в исходном коде. Эта задача является NP-сложной. В работе рассматривается метод поиска оптимального сдвига старта последовательностей через быстрое преобразование Фурье для различных вариантов постановки задачи (числа последовательностей). Цепочки инструкций представляются в виде временных рядов использования ресурсов (исполнительных портов) и регистров вычислительного ядра. При работе с регистрами ряды взаимной корреляции строятся через преобразование Фурье размерностью K, где K – число рядов. При работе с портами каждый ряд разделяется на несколько каналов, соответствующих различной нагрузке порта, для каждого канала выполняется преобразование Фурье размерностью K–1, далее результаты суммируются с заданными весами. Показано, что при K=3 в обоих случаях частотное представление имеет меньшую вычислительную сложность, чем прямой перебор. Метод также подходит для поиска оптимальной точки разреза последовательности с целью конвейерного исполнения в рамках одного потока. Алгоритм успешно находит точки оптимального совмещения для набора тестовых последовательностей с высокой дисперсией локальных средних интенсивностей использования индивидуального ресурса и сочетаний ресурсов.

Ключевые слова: быстрое преобразование Фурье, взаимная корреляция, планирование инструкций, регистровое давление, исполнительные порты, слияние потоков, оптимизация компилятора.

Для цитирования: Буевич Е.А. Фазовая оптимизация однопоточного выполнения независимых программных последовательностей. *Моделирование, оптимизация и информационные технологии.* 2026;14(7). URL: https://moitvivt.ru/ru/journal/article?id=2456 DOI: 10.26102/2310-6018/2026.58.7.009

Phase optimization of single-threaded execution of independent program sequences

E.A. Buevich

Moscow State Technological University "STANKIN", Moscow, the Russian Federation

Abstract. This paper considers one aspect of the joint execution of several data-independent program sequences in a single thread, achieved by interleaving their instructions in the source code. This problem is NP-hard. The paper considers a method for finding the optimal sequence start shift using the fast Fourier transform for various problem formulations (number of sequences). Instruction chains are represented as time series of resource (execution ports) and core register utilization. When working with registers, cross-correlation series are constructed using the Fourier transform of dimension K, where K is the number of series. When working with ports, each series is divided into several channels corresponding to different port loads. A Fourier transform of dimension K–1 is performed for each channel, and the results are then summed with the specified weights. It is shown that for K=3, in both cases, the frequency representation has lower computational complexity than brute force. The method

is also suitable for finding the optimal sequence cut point for pipelined execution within a single thread. The algorithm successfully finds optimal matching points for a set of test sequences with high variance of local average usage intensities of individual resources and combinations of resources.

Keywords: fast Fourier transform, cross-correlation, instruction scheduling, register pressure, execution ports, thread fusion, compiler optimization.

For citation: Buevich E.A. Phase optimization of single-threaded execution of independent program sequences. *Modeling, Optimization and Information Technology.* 2026;14(7). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2456 DOI: 10.26102/2310-6018/2026.58.7.009

Введение

Задача оптимального совмещения потоков близка проблемам слияния циклов (loop fusion) и программной конвейеризации. Одним из условий для ее постановки является возможность представления нескольких участков кода в виде ациклических графов. В случае слияния циклов подобным участком является тело цикла, совмещаемое с одной или несколькими итерациями другого. В случае конвейеризации одна из частей программы совмещается с другой через разнесение зависимых данных по разным итерациям.

Второе необходимое условие для подобной оптимизации совпадает с условием применимости объединения циклов – участки кода должны находиться в одном узле графа зависимостей по управлению (Control Dependency Graph, CDG [1]) или в эквивалентных. Наиболее точная формулировка эквивалентного положения узлов в графе управления приведена в документации к компилятору LLVM[1]: «Узлы эквивалентны в потоке управления, если один из них является доминатором для второго, а второй – постдоминатором для первого».

Зависимости по данным между участками, если они существуют, также должны соблюдаться или трансформироваться, но в рамках данной работы для упрощения предполагается их отсутствие. Это характерно для конвейеризованного кода, предполагающего итеративное выполнение.

В современных компиляторах (gcc, llvm) используется алгоритм модульного планирования с качанием (Swing Modulo Scheduling (SMS), работающий в масштабе инструкций. Он позволяет построить близкое к оптимальному расписание выполнения в небольших масштабах, но со смешиванием длительных последовательностей справляется плохо [2, 3]. Для подобных задач существует промежуточное представление кода MLIR [4], однако его эвристики строятся от алгоритма, а не от инструкций.

Целью данной работы является разработка метода поиска оптимального с точки зрения времени исполнения в одном потоке совмещения длинных последовательностей инструкций.

Для этого потребность каждого потока в определенном ресурсе представляется в виде нескольких рядов. Способ представления различается для равнозначных (регистры) и неравнозначных (исполнительные порты) ресурсов. Далее с помощью быстрого преобразования Фурье строятся ряды взаимной корреляции, от элементов которых подсчитывается целевая функция.

Приведенный алгоритм может использоваться как для поиска точек разреза длинного цикла для последующей конвейеризации, так и для оптимального слияния уже имеющихся последовательностей. По области применения его можно отнести к алгоритмам глобальной оптимизации.

[1] Barton K., Doerfert J., Finkel H., Kruse M. *Revisiting Loop Fusion and its place in the loop transformation framework.* LLVM Developer Meeting. URL: https://llvm.org/devmtg/2018-10/slides/Barton-LoopFusion.pdf (дата обращения: 24.05.2026).

Материалы и методы

Рассмотрим совмещение в одном потоке двух программных последовательностей (цепочек) длиной m и n тактов соответственно и экстраполируем полученные результаты на добавление третьей и последующих цепочек, а также на попарное сравнение набора последовательностей.

Примем, что последовательность из m тактов более длинная. Если ограничиться только равномерным смешиванием, всего возможно $m+n$ вариантов их взаимного расположения. Обозначим суммарную длину всех смешиваемых последовательностей как $T = m + n + ...$

Равнозначные ресурсы (регистры). Возможность одновременного выполнения двух блоков кода ограничивается их потребностью в регистрах. При превышении доступного числа регистров K процессору придется выполнять дополнительные обращения к памяти из-за взаимного вытеснения (register spilling). При этом в стек вытесняются регистры с максимальным временем до следующего обращения (время повторного использования, reuse distance, RD). Последовательность инструкций может быть охарактеризована через среднюю величину повторного обращения $E[RD]$, как отношение суммы повторных обращений к их количеству.

Прямой перебор в данном случае – это самый точный метод подсчета вытеснений регистров, поскольку позволяет не учитывать одно и то же вытеснение на нескольких последовательных тактах. Для этого, для каждого такта каждой последовательности использование регистров должно быть описано вектором $\overline{Q_d}(t) = \{RD(k,t) - t\}$, хранящим разницу между следующим использованием каждого регистра k и временем t или определенное большое значение, если регистр не используется. При обнаружении вытеснения счетчик вытеснений увеличивается, а регистр с наибольшим $Q_{d,k}(t)$ считается пустым в соответствующей значению $Q_{d,k}(t)$ последовательности до момента повторного использования.

Если принять, что для участков, на которых присутствует только одна последовательность, вытеснение невозможно, сложность полного перебора равна $N \cdot M$ операций.

Для оценки количества вытеснений регистров с помощью быстрого преобразования Фурье (БПФ) используется геометрическое представление в виде единичных матриц $A_{i,KxT}$. В данном случае удобнее будет величина интенсивности использования $RP(k,t) = 1/Q_{d,k}(t)$ (Reuse Probability в зарубежных статьях, [5, 6]), обратная расстоянию до следующего использования регистра k, или равная нулю, если он больше не используется. Для каждой точки кода каждой последовательности создается вектор $\overline{Q_p}(t)$, содержащий ненулевые величины $RP(r,t)$ в порядке убывания:

$$\overline{Q_p}(t) = \{RP(r_{i_1},t), RP(r_{i_2},t), \dots, RP(r_{i_k},t) | RP(r_{i_1},t) \geq RP(r_{i_2},t) \geq \dots \geq \geq RP(r_{i_k},t), RP\left(r_{i_j},t\right) > 0\}.$$

На основании этих векторов строятся матрицы использования регистров. Для первой последовательности:

$$A_1(k,t) = \mathbb{I}_{(\|\overline{Q_{p,1}(t)}\| > k)},$$

где $\mathbb{I}_{(\|\overline{Q_{p,1}(t)}\| > k)}$ – индикаторная функция, равная 1, если используется более чем k регистров, и 0 – в противоположном случае. Аналогично для второй последовательности:

$$A_2(k,t) = \mathbb{I}_{(\|\overline{Q_{p,2}(t)}\| \geq K-k)}.$$

Матрица второй последовательности составляется отраженной по оси времени. Их общий вид показан на Рисунке 1.

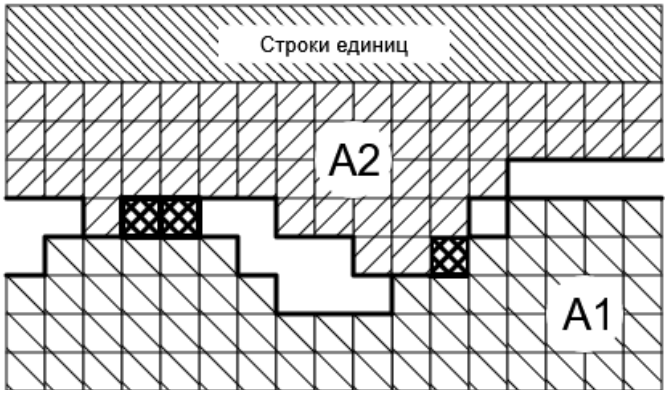

Рисунок 1 – Матрицы использования равнозначных ресурсов
Figure 1 – Matrices of utilization of equivalent resources

К получившимся матрицам применяется упрощенный алгоритм пространственной спектральной упаковки [7]. Каждая матрица переводится в частотную область через двумерное быстрое преобразование Фурье:

$$\tilde{A}_j(u,v) = \mathcal{F}_2\{A_j\} = \sum_{t=0}^{T-1}\sum_{k=0}^{K-1} A_j(k,t)\cdot e^{-i2\pi(\frac{u\cdot k}{K}+\frac{v\cdot t}{T})}. \quad (1)$$

Частотная матрица взаимной корреляции находится через комплексно-сопряженное умножение:

$$\tilde{B}(u,v) = \tilde{A}_1(u,v)\cdot \tilde{A}_2{}^*(u,v), \quad (2)$$

где $\tilde{A}_2{}^*(u,v)$ – комплексно-сопряженная матрица для $\tilde{A}_2(u,v)$.

Обратный переход к временной области:

$$R(k,z) = Re\{\mathcal{F}_2{}^{-1}\{\tilde{B}\}\} = Re\left\{\frac{1}{KT}\sum_{v=0}^{T-1}\sum_{u=0}^{K-1}\tilde{B}(u,v)\cdot e^{i2\pi(\frac{u\cdot k}{K}+\frac{v\cdot z}{T})}\right\}. \quad (3)$$

Верхней строкой результата ($k = 0$) будет количество превышений требуемого числа регистров над доступным для данного сдвига z. Минимальное значение будет соответствовать оптимальному сдвигу: $F(z) = R(0,z)$, где $F(z)$ – целевая функция оптимизации. Сложность подобного вычисления составляет $3TK\cdot\log_2 TK + TK$ операций.

Поскольку остальные строки не нужны, можно применить усеченное БПФ [8] для обратного преобразования:

$$F(z) = Re\left\{\frac{1}{KT}\sum_{v=0}^{T-1}\left(\sum_{u=0}^{K-1}\tilde{B}(u,v)\right)\cdot e^{i2\pi\frac{v\cdot z}{T}}\right\}. \quad (4)$$

Здесь каждый столбец матрицы B суммируется, и к результату применяется одномерное преобразование. Эта оптимизация снижает общее число операций до $2TK\cdot\log_2 TK + + T(\log_2 T + 1)$.

Для двух рядов инструкций данный метод проигрывает как по точности из-за невозможности учета повторных вытеснений, так и по скорости для размеров последовательностей, встречающихся на практике (из-за добавления второго измерения и более высокой константы). Однако ситуация меняется как при попарном сравнении некоторого набора последовательностей, так и при добавлении третьей последовательности.

Выигрыш при попарном сравнении достигается за счет того, что основная нагрузка ($2TK \cdot \log TK$) приходится на перевод матриц в частотную область, а вычисление результата собственно объединения выполняется за $T(\log T + 1)$ операцию, в то время как прямой перебор в каждом случае выполняется полностью.

Также данный алгоритм может быть модифицирован для слияния трех последовательностей при помощи трехмерного БПФ. В этом случае матрицы составляются следующим образом (размеры последовательностей равны n_1, n_2 и n_3, суммы длин $T_2 = n_1 + n_2$ и $T_3 = n_1 + n_3$):

$$A_1(k,t,t) = \mathbb{I}_{\{\|\overline{Q_{p,1}(t)}\|>k\}}, t = 0..n_1 - 1, \quad (5)$$

$$A_2(k,t,t_2) = \mathbb{I}_{(\|\overline{Q_{p,2}(t)}\|\geq K-k)}, t = 0..n_2 - 1, t_2 = 0..T_2 - 1,$$

$$A_3(k,t_2,t) = \mathbb{I}_{(\|\overline{Q_{p,2}(t)}\|\geq K-k)}, t = 0..n_3 - 1, t_2 = 0..T_3 - 1.$$

При использовании трехмерного преобразования и оптимизации, аналогичной (4), результатом станет двумерная матрица размером $T_2 \times T_3$. После нахождения в ней минимального элемента с индексами x_{min} и y_{min}, сдвиги второй и третьей последовательности относительно первой можно получить по следующим формулам [8]:

$$z_2 = \begin{cases} x_{min}, x_{min} < n_2 \\ x_{\min} - T_2, x_{min} \geq n_2 \end{cases},$$

$$z_3 = \begin{cases} y_{min}, y_{min} < n_3 \\ y_{\min} - T_3, y_{min} \geq n_3 \end{cases}.$$

Сложность этого вычисления составит:

$$3T_2T_3K(\log_2(T_2T_3K) + 1) + T_2T_3\log_2(T_2T_3),$$

что для невысоких K значительно выгоднее прямого перебора, учитывающего RP или RD вытесняемого регистра, для которого сложность в данном случае равна:

$$n_1n_2n_3 + n_1n_2(n_1 + n_2) + n_1n_3(n_1 + n_3) + n_2n_3(n_2 + n_3).$$

Так для трех рядов из 500 элементов и 16 регистров 3D БПФ примерно в 5 раз быстрее, чем полный перебор.

Для четырех и более последовательностей из-за экспоненциального роста объема пространства выигрыш теряется.

Точность при сравнении двух матриц может быть также существенно увеличена. Для этого в матрицу i, соответствующей последовательности с большим $E[RD]$, вместо единиц записываются непосредственно элементы $\overline{Q_p(t)}$:

$$A_1(k,t) = Q_{p,1,k}(t) \text{ или } A_2(k,t) = Q_{p,1,K-k-1}(t).$$

Это значительно снижает неточность, вызванную «повторными» вытеснениями, уменьшая максимальную ошибку с RD до $\sim \ln RD$.

Хотя компиляторы стремятся оптимизировать используемое число регистров (перемещение кода с учетом регистрового давления, [5]), число интенсивно используемых регистров часто ограничено невысокой латентностью инструкций. В силу этого единичные взаимные вытеснения не оказывают заметного влияния на итоговую производительность.

Неравнозначные ресурсы. Моделировать использование портов несколько сложнее, поскольку часть инструкций требует для своего выполнения конкретные исполнительные порты (BMI, векторные перестановки в процессорах Intel), в то время как другие допускают выполнение на большом множестве (скалярная арифметика). В

данном случае используется дискретная массовая функция вероятности использования порта потоком в заданном такте. Сама случайная величина имеет Пуассон-биномиальное распределение, поскольку является суммой некоторого количества независимых испытаний Бернулли с разной вероятностью успеха. В каждом отдельном испытании она выбирается из фиксированного набора, зависящего от процессора. Для Intel это 1,1/2,1/3,1/4.

Для нахождения оптимального временного сдвига необходимо минимизировать математическое ожидание превышения порога (для исполнительного порта порог равен 1). В теории анализа рисков и стохастической оптимизации данная величина эквивалентна показателю ожидаемого дефицита [9]. Целевая функция представляет собой интегральный ожидаемый штраф за выход случайной величины за пределы ограничений:

$$F(H) = E[\max 0, X(t) - H] = \sum_{x>H}(x - H)P(X(t) = x). \quad (6)$$

В данном случае существенно упростить расчеты помогает ограниченная пропускная способность декодера вычислительного ядра – область определения функции $D(P(z))$ невелика, например, для процессоров Intel Skylake она равна 4. Исходя из этого возможно численно рассчитать вероятность превышения заданного порога в каждом конкретном такте, как свертку распределений:

$$P(Z(t) = z) = \sum_k p_1(t, k) \cdot p_2(t, z - k), \quad (7)$$

где p_1 и p_2 – массовые функции вероятности нагрузки порта для момента времени t смешиваемых последовательностей.

Формул (6) и (7) достаточно для подсчета потерь алгоритмом квадратичной сложности. Однако, как и в предыдущем случае, для нахождения сдвигов при объединении большего количества последовательностей, частотное представление дает преимущество в скорости.

Для использования быстрого преобразования Фурье каждый ряд представляется как $K = |X|$ рядов вероятностей (мощность множества размера нагрузки), таким образом получается $2K$ рядов для двух последовательностей ($j \in \{1,2\}$).

$$A_{j,k}(t) = \sum_{y=0}^{K-1} p_j(t, y) \cdot \mathbb{I}_{(y=k)}.$$

Ряды переводятся в частотную область:

$$\tilde{A}_{j,k}(\omega) = \mathcal{F}\{A_{j,k}\} = \sum_{t=0}^{T-1} A_{j,k}(t) \cdot e^{-i2\pi(\frac{\omega \cdot t}{T})}.$$

Создается $2K - 1$ спектров сумм $\tilde{B}_s$ для каждого возможного значения нагрузки и выполняется суммирование всех комплексно-сопряженных произведений частотных рядов первой последовательности на частотные ряды второй с добавлением результата к соответствующему $\tilde{B}_s$:

$$\tilde{B}_s(\omega) = \tilde{B}_s(\omega) + \tilde{A}_{1,k_1}(\omega) \cdot \tilde{A}_{2,k_2}(\omega), s = k_1 + k_2.$$

Используя свойство линейности БПФ [10], можно собрать целевую функцию непосредственно в фазовом пространстве, и, таким образом, уменьшить количество обратных преобразований:

$$\tilde{B}(\omega) = \sum_{s=H+1}^{2K-2} \tilde{B}_s(\omega) \cdot (s - H). \quad (8)$$

Тогда целевая функция выглядит как:

$$F(z) = Re\{\mathcal{F}^{-1}\{\tilde{B}\}\} = Re\left\{\frac{1}{T}\sum_{=0}^{T-1} \tilde{B}(\omega) \cdot e^{i2\pi(\frac{\omega \cdot z}{T})}\right\}. \quad (9)$$

Поскольку количество рядов в БПФ компенсируется операцией свертки для прямого перебора, в данном случае использование частотного представления лучше практически для любой постановки задачи. Итоговая сложность алгоритма составляет $(KT+1)\log_2 T + K^2T$, что значительно лучше, чем K^2T^2 при лобовом поиске. Для рядов из 500 элементов при $K = 4$ метод, основанный на БПФ, быстрее примерно в 500 раз.

Еще одним достоинством использования частотного представления является тот факт, что поиск корреляций использования портов строго в масштабах такта имеет мало практической пользы. Существует достаточно факторов, смещающих последовательности на несколько тактов вперед или назад, в том числе коллизии на ресурсах. Намного практичнее рассчитывать эти величины в плывущем окне. Целевая функция подсчитывает пересечения повышенной нагрузки на ресурс в заданном временном интервале, оставляя планировщику компилятора и внеочередному исполнению процессора разрешение коллизий в масштабе такта. В частотном пространстве такой подход практически не приводит к увеличению вычислительной сложности алгоритма, поскольку достаточно умножения спектра результата на спектр окна шириной w:

$$W(t) = \begin{cases} \frac{1}{w}, t < w \\ 0, t \geq w \end{cases},$$

$$\widetilde{W}(\omega) = \mathcal{F}\{W(t)\} = \sum_{t=0}^{T-1} W(t) \cdot e^{-i2\pi(\frac{\omega * t}{T})}.$$

Формула (8) модифицируется к виду:

$$\tilde{B}(\omega) = \widetilde{W}(\omega) \cdot \sum_{s=H+1}^{2K-2} \tilde{B}_s(\omega) \cdot (s - H).$$

Данный алгоритм легко масштабируется в двумерное БПФ для трех рядов, аналогично предыдущему пункту.

Целевая функция для всех ресурсов при поиске точки смещения для двух последовательностей выглядит как:

$$F(z) = k_l \cdot max\{0, z - m, n - z\} + \sum_J k_J F_J(z), \quad (10)$$

где $F_J(z)$ – целевая функция для соответствующего ресурса, полученная по формулам (4) или (9), z – временной сдвиг, k_l – коэффициент штрафа за выход более короткой за пределы более длинной. Поскольку для всех типов ресурсов целевые функции оперируют тактами, $k_l = 1$. Коэффициенты k_J для портов также равны 1, для регистров равны оценке в тактах последствий вытеснения данных из регистров (замена части инструкций на работающие напрямую с памятью).

Целевая функция при поиске точки разделения последовательности длиной m на две с последующим совмещением (в этом случае оба набора матриц создаются из исходной последовательности) выглядит как:

$$F(z) = k_l \cdot |m - 2z| + \sum_J k_J F_J(z). \quad (11)$$

Результаты и обсуждение

Исходный код реализации данного алгоритма доступен по ссылке https://github.com/evgnort/merger. Программа принимает на вход два файла в формате ассемблера AT&T и выводит имя файла номер строки, в которой должно начаться выполнение другого файла. Тестовые последовательности находятся в директории /samples и имеют имена sampleNa.s и sampleNb.s, где N номер теста. Общей чертой всех

тестовых последовательностей является дисбаланс потребности в определенном ресурсе в различных участках кода.

sample1: Первая последовательность является оптимизированным векторным кодом умножения матриц, а вторая содержит сборку буфера из чередующихся строк с последующим сохранением результата. Данный пример взят из кода библиотеки линейной алгебры.

Результат:

– последовательно: 34 такта;

– лучшее совмещение: 34 такта (sample1a.s line 30) (ports: 1 + regs: 0 + lenght: 10).

Алгоритм смог совместить только конечный участок одной последовательности и начальный второй из-за интенсивного использования векторных регистров, что говорит о хорошей оптимизации кода.

sample2: В данном случае одна из последовательностей содержит много тяжелых инструкций с высокой латентностью, однако обе последовательности интенсивно используют векторные регистры.

Результат:

– последовательно: 279 тактов;

– лучшее совмещение: 167 тактов (sample2a.s line 9) (ports: 20 + regs: 20 + lenght: 0).

Алгоритм нашел возможность совмещения, несмотря на геометрическое превышение количества регистров на четыре единицы.

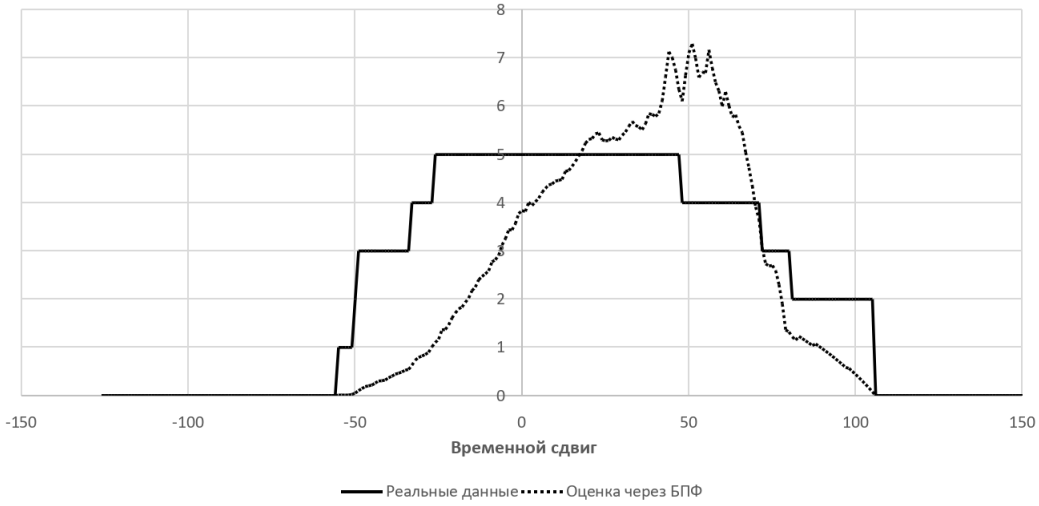

Рисунок 2 – Количество вытеснений регистров для последовательностей sample2
Figure 2 – Number of register spillings for sample2 sequences

На Рисунке 2 показано реальное количество вытеснений регистров и оценка через быстрое преобразование Фурье. Коэффициент корреляции Пирсона составляет 0,81. На некоторых участках первая производная ошибки достигает половины штрафа за единичный сдвиг, что оказывает ограниченное влияние на выбор точки совмещения (сдвиг оптимального положения на 5 тактов для данных последовательностей). Подобное приближение вытеснения регистров требует дополнительного исследования.

sample3: Обе последовательности интенсивно работают с памятью.

Результат:

– последовательно: 136 тактов;

– лучшее совмещение: 122 такта (sample3a.s line 3) (ports: 51 + regs: 0 + lenght: 1).

В этом случае практически все варианты совмещения с полным наложением дают близкие результаты.

Заключение

Приведенные результаты показывают, что при близкой точности для определения штрафа за вытеснение регистров при совместном исполнении в одном потоке использование частотного представления имеет преимущество над перебором только для некоторых случаев постановки задачи. В то же время при вычислении штрафов за перегрузку исполнительных портов этот подход значительно превосходит перебор по скорости, не уступая в точности. Данный алгоритм может использоваться для глобальной оптимизации кода при обнаружении участков-кандидатов на совмещение при помощи других эвристик.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Ferrante J., Ottenstein K.J., Warren J.D. The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*. 1987;9(3):319–349.
2. Llosa J., González A., Ayguadé E., et al. Swing Modulo Scheduling: A Lifetime-Sensitive Approach. In: *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Technique, 20–23 October 1996, Boston, MA, USA*. IEEE; 1996. P. 80–91. https://doi.org/10.1109/PACT.1996.554030
3. Zhang Z., Liu B. SDC-based modulo scheduling for pipeline synthesis. In: *Proceedings of the 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 18–21 November 2013, San Jose, CA, USA*. IEEE; 2013. P. 211–218. https://doi.org/10.1109/ICCAD.2013.6691121
4. Lattner C., Amini M., Bondhugula U., et al. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In: *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 27 February – 3 March 2021, Seoul, Korea (South)*. IEEE; 2021. P. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308
5. Gupta R., Bodík R. Register Pressure Sensitive Redundancy Elimination. In: *Proceedings of the 8th International Conference, CC'99, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99, March 22–28 1999, Amsterdam, Netherlands*. Berlin, Heidelberg: Springer; 1999. P. 107–121. https://doi.org/10.1007/978-3-540-49051-7_8
6. Yang Y., Xiang P., Kong J., et al. A GPGPU compiler for memory optimization and parallelism management. In: *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 5–10 June 2010, Toronto, Ontario, Canada*. New York: Association for Computing Machinery; 2010. P. 86–97. https://doi.org/10.1145/1806596.1806606
7. Cui Q., Rong V., Chen D., et al. Dense, Interlocking-Free and Scalable Spectral Packing of Generic 3D Objects. *ACM Transactions on Graphics*. 2023;42(4):1–14. https://doi.org/10.1145/3592126
8. Sorensen H.V., Burrus C.S. Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*. 1993;41(3):1184–1200. https://doi.org/10.1109/78.205723
9. Perreault-Lafleur C., Carvalho M., Desaulniers G. A stochastic integer programming approach to reserve staff scheduling with preferences. *International Transactions in Operational Research*. 2025;32(1):289–313. https://doi.org/10.1111/itor.13298
10. Cooley J.W., Tukey J.W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*. 1965;19(90):297–301. https://doi.org/10.2307/2003354

ИНФОРМАЦИЯ ОБ АВТОРЕ / INFORMATION ABOUT THE AUTHOR

Буевич Евгений Андреевич, аспирант, Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация.
e-mail: gftcompmod@gmail.com

Evgeniy A. Buevich, Postgraduate, Moscow State Technological University "STANKIN", Moscow, the Russian Federation.

Статья поступила в редакцию 27.05.2026; одобрена после рецензирования 30.06.2026; принята к публикации 16.07.2026.

The article was submitted 27.05.2026; approved after reviewing 30.06.2026; accepted for publication 16.07.2026.