БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

МАРИЯ РУМЕНОВА ПАШИНСКА-ГАДЖЕВА

ОПТИМИЗАЦИЯ И ПАРАЛЕЛИЗАЦИЯ НА АЛГОРИТМИ, СВЪРЗАНИ С ТЕОРИЯ НА КОДИРАНЕТО

Автореферат
на дисертация

за присъждане на образователна и научна степен
"доктор"

Професионално направление:
4.6 Информатика и компютърни науки
Научна специалност:
Информатика

Научен ръководител:
проф. дмн Илия Буюклиев

Велико Търново
2024

Увод

В текущата работа са разгледани подходи за оптимизация и паралелизация на алгоритми от теория на кодирането. Един от основните въпроси, засегнати тук, е свързан с избора на метод за паралелизация в зависимост от задачата. Задачите, които се решават, са важни както в теоретичен, така и в практически аспект. В практическо отношение теория на кодирането е свързана с комуникационни и компютърни технологии, отнасящи се до защита на информацията, компресиране, архивиране, автентикация и съхранение на данните, хеширане, блокчейн и други [12, 13, 26].

Голяма част от изследваните проблеми изискват използването на паралелни изчисления за тяхното решаване [4, 5, 18, 35]. За целта е необходимо добро познаване на различните технологии и интерфейси, използвани за паралелни изчисления, за да бъде възможно оптималното им прилагане. Съвременните изчислителни технологии се развиват в много направления. Едно от основните е разработването на многопроцесорни и многоядрени архитектури, които позволяват извършване на изчисления едновременно. Друго направление за развитие на изчислителната техника е в интегриране на архитектури, предназначени за специфични задачи в системи за общи изчисления. Такива са ускорителите, които се характеризират с голям брой изчислителни единици [24, 29]. Системите, които в допълнение към централен процесор със стандартна архитектура използват и ускорител, се наричат хетерогенни. В следствие на това развитие, паралелните изчислителни системи стават все по-достъпни. В последните десетилетия компютърните архитектури се развиват, като се добавят повече ядра (изчислителни единици) за един процесор и повече процесори за дадена система. Във всяко изчислително ядро на централните процесори също така се добавят и разширени векторни регистри. Всяка от представените характеристики на паралелните архитектури е обект на самостоятелно и обстойно изучаване. В текущата работа по-обстойно ще бъде коментиран специфичен подход за паралелизация, който използва тези разширени регистри. Такъв подход се нарича векторизация. Тя се изразява в изпълнение на операции върху множество от елементи едновременно. Разработени са регистри с различни дължини, кратни на 128, като за работа с тях са създадени и допълнителни инструкции за централните процесори. Векторизацията чрез разширени регистри може да бъде извършена автоматично от компилатора или като се използват специални функции, разработени за езиците C/C++. Тези функции заедно с нови дефинирани типове от данни са включени в библиотеки и позволяват лесното използване на разширените регистри без изрично позна-

ние на асемблерските инструкции. Този подход също така дава възможност за използването на други паралелни интерфейси в комбинация с векторизацията, като така предоставя допълнително ускорение.

При векторизация, бързодействието, което се постига, зависи от следните основни фактори:

- използваните инструкции - инструкциите за централните процесори условно могат да бъдат разглеждани като леки (побитови операции, сравнение, събиране и др.) и тежки (съчетаване, пермутации и др.). Леките инструкции се изпълняват приблизително толкова бързо, колкото стандартна инструкция (често те се изпълняват за един такт на централния процесор), за разлика от тежките. Тази категоризация зависи от хардуерната имплементация.

- коефициент на векторизация - показва максималния брой на координатите на даден вектор, които могат да бъдат записани в даден регистър [2]. Този коефициент зависи от големината на регистъра и начина на представяне на елементите в паметта.

- коефициент на използване - показва каква част от регистъра е използвана при изчисленията. Стойността му е между 0 и 1, като при 1 се използва целият регистър, при 1/2 - половината, а при 0 не се използва регистър. Този коефициент зависи от представянето на елементите и дължината на кода, за който се извършват изчисленията.

Тези параметри се използват при анализирането на ефективността на реализацията на представените алгоритми.

В текущата дисертация се разглеждат линейни кодове и векторизация на свързани с тях алгоритми. Решаването на основните задачи в теория на кодирането включва много и различни алгоритмични проблеми, като намиране на тегловни инварианти (тегловен спектър, кодови думи с фиксирано тегло и др.) на линеен код. Основната цел на дисертацията е разработване на оптимизирани алгоритми за изчисление на тегловни инварианти на линейни кодове, които са базирани на тегловното разпределение на кода. Те са част от много алгоритми за решаване на задачите за генериране и класификация на кодове. За целта е разработена оптимизирана библиотека **LinCodeWeightInv**, като е използвана векторизация. Библиотеката включва интерфейсна част, модул за тестване и верификация и пълно описание на включените функционалности. Част от алгоритмите също така са включени в софтуерния пакет **QExtNewEdition**. Тези софтуерни пакети са използвани за изучаването на фамилии от кодове,

свързани с двоични линейни кодове, достигащи границата на Грей-Ранкин.

В текущата работа са разгледани следните основни цели, задачи и начините за решаването им:

- Векторизирана реализация на алгоритми за намиране на тегловен спектър на линеен код над полета с до 64 елемента, включително. За имплементацията са разработени два основни типа от алгоритми - оптимизирани алгоритми от високо ниво, които описват подхода за генериране на нова кодова дума и векторизирани алгоритми от ниско ниво, които реализират операциите събиране на вектори и намиране на теглото на вектор. Основната оптимизация в алгоритмите от високо ниво се състои в генерирането само на непропорционални кодови думи, като всяка нова кодова дума се получава единствено чрез събиране на вектори. Алгоритмите от ниско ниво включват в себе си различни реализации на функции за събиране на вектори и намирането на теглото на вектор, като се използват специални инструкции и разширени векторни регистри. Разработените функции се различават в зависимост от броя на елементите в полето и избрания набор от разширени инструкции за централния процесор. Анализирана е ефективността на имплементираните алгоритми в x86 архитектури, като се използват регистри с дължина 128 и 256 бита и съответните инструкции, предназначени за работа с тях. Имплементациите са сравнени с функции за намирането на тегловното разпределение в широко използваните пакети за компютърна алгебра Magma и GAP. Също така е анализирано влиянието на компилатора при използване на директна векторизация.

- Разширяване на основните алгоритми от ниско ниво за работа над прости полета с по-малко от 128 елемента. Разгледано е представяне на елементите на полето в паметта, като се използва беззнаков тип от данни с големина 8 бита. Това представяне позволява изчисленията да бъдат извършени над по-големи прости полета, без да се излиза от обхвата на стойности за дадения тип от данни. Това е възможно, тъй като се използва единствено операцията събиране на вектори и наличието на т.нар. функции със сатурация. Функциите със сатурация за разширените векторни регистри позволяват при излизане от обхавата на дадения тип от данни да бъде записана валидна стойност (минималната или максималната за дадения тип в зависимост от резултата) вместо изрязване на част от стойността или грешно интерпретиране на побитовото представяне на резултата. Разработените алгоритми са имплементирани чрез разширени-

те набори от инструкции SSE4.1 и AVX512 за x86 архитектури и NEON набор от регистри за ARM архитектури. За реализиране на алгоритмите са изучени основните характеристики на инструкциите от вида AVX512 и NEON. Анализирана е ефективността на различните реализации, като е използван алгоритъм за намиране на тегловен спектър на линеен код.

- Анализиране на връзките между фамилии от кодове с две и три тегла, свързани със самодопълнителни кодове, които достигат границата на Грей-Ранкин. За целта са дефинирани четири фамилии от кодове с две тегла и две фамилии от кодове с три тегла. Описани са връзките между дефинираните фамилии, като са представени конструкции за получаване на кодове от различните фамилии при даден код от коя да е от останалите фамилии от кодове. Описани са и връзките им със самодопълнителни кодове, достигащи границата на Грей-Ранкин. Представени са конструкции за построяване на кодове с две тегла от описаните фамилии с размерност $k+2$ при зададен код от съответна фамилия с размерност k. Тези връзки между кодовете са използвани за конструиране на самодопълнителни кодове с параметри $[120, 9, \{56; 64; 120\}]$.

- Разработване на оптимизирана и преносима библиотека за намиране на тегловни инварианти на линейни кодове над полета $\mathbb{F}_q$, където $q \leq 64$. Библиотеката включва шест основни интерфейсни функции, за които са разработени три различни начина за въвеждане на данните. Създадени са два основни модула за използване на библиотеката - интерфейсен модул и модул за тестване и верификация. Разработената библиотека работи за x86 и ARM архитектури и различни набори от регистри. За имплементирането на библиотеката са изучени различни платформи, които се използват за създаване на софтуер и изготвяне на пълна документация.

Глава 1

В тази глава са представени някои основни дефиниции и твърдения в теория на кодирането, както и архитектури и подходи за паралелизация на алгоритми. В изложението се следват монографиите [25, 30, 32]. Нека $\mathbb{F}_q$ е крайното поле с q елемента, а $\mathbb{F}_q^n$ - n-мерното векторно пространство над полето. *Разтояние по Хеминг* между два вектора $x = (x_1, \ldots, x_n)$ и $y = (y_1, \ldots, y_n)$ се нарича броят на координатите, в които те се различават. Означава се с $d(x, y) = |\{i | x_i \neq y_i\}|$. *Тегло по Хеминг* на вектор $x = (x_1, \ldots, x_n)$ се нарича броят на ненулевите координати на вектора. Означава се с $wt(x) = |\{i | x_i \neq 0\}|$.

Линеен код с дължина n, размерност k над крайно поле с q елемента се нарича всяко k-мерно подпространство на n-мерното векторно пространство $\mathbb{F}_q^n$. Елементите на кода се наричат *кодови думи*, а параметрите n и k се наричат съответно *дължина* и *размерност* на кода. Матрица с k реда и n стълба, чиито редове образуват базис на кода C, се нарича *пораждаща матрица* за кода. Един код е *проективен*, ако не съдържа нулеви и пропорционални координати.

Един основен параметър на линеен код е неговото *минимално разстояние* $d(C)$, което е най-малкото от всички разстояния между две различни кодови думи. *Минималното тегло* на линеен код се нарича най-малкото от всички ненулеви тегла в кода. За линеен код C минималното тегло и минималното разстояние съвпадат. Линеен код C с дължина n, размерност k и минимално разстояние d се означава с $[n, k, d]$. Остатъчен код на даден код C по отношение на кодова дума $x \in C$ се нарича код $Res(C, x)$, който се получава от рестрикцията на C върху нулевите координати на кодовата дума x. Редицата $(A_0, A_1, \ldots, A_n)$, където A_i е броят на кодовите думи с тегло i, се нарича *тегловен спектър* на линейния код. За всеки линеен $[n, k, d]$ код е в сила $A_0 = 1$ и $A_1 = A_2 = \cdots = A_{d-1} = 0$. Също така се вижда, че $A_0 + A_1 + \cdots + A_n = q^k$. Два линейни $[n, k, d]$ кода над поле $\mathbb{F}_q$ се наричат еквивалентни, ако всички кодови думи на единия код могат да се получат от кодовите думи на другия чрез последователност от следните трансформации:

- пермутация на координатите;
- умножение на елементите в дадена координата с ненулев елемент на $\mathbb{F}_q$;
- прилагане на автоморфизъм на полето към елементите във всички координатни позиции.

Релацията на еквивалентност разделя множеството на всички кодове с дадени параметри на класове на еквивалентност. Задачата за класифициране на линейни кодове се състои в намирането на представител на всеки клас на еквивалентност.

Някои от методите и алгоритмите за генериране и класификация на линейни кодове използват целия или частичен спектър на кода. Еквивалентни кодове имат равни минимални разстояния и тегловни спектри, тъй като позволените трансформации запазват теглото по Хеминг. Това прави задачата за намиране на тегловен спектър на линеен код основна в теория на кодирането. Доказано е, че задачата е NP-пълна [3].

Задачите за конструиране, класификация, намиране на тегловен спектър

в общия случай изискват голям изчислителен ресурс. Тогава оптимизацията и паралелизацията на алгоритмите са важни за решаването на дадените задачи в обозримо време. Повече за оптимизацията на програмен код на C/C++ и някои популярни техники за паралелизация може да се намери в [20, 21, 31, 33, 34, 36].

Векторизацията е сред най-подходящите подходи за паралелизация на алгоритми, свързани с изучаване на линейни кодове, тъй като основните изчисления са свързани с операции над вектори. Векторизацията в съвременните процесори се базира на използване на разширени регистри, чиято дължина е кратно на дължината на една компютърна дума. При векторизация, бързодействието, което се постига, зависи от следните основни фактори:

- използваните инструкции (леки и тежки);
- коефициент на векторизация [2];
- коефициент на използване.

Тези параметри се използват при анализирането на ефективността на реализацията на представените алгоритми.

За работа с разширените регистри са разработени допълнителни типове от данни и инструкции. Инструкциите за 128-битови регистри на централните процесори с x86 архитектури са известни като Streaming SIMD Extensions (SSE). Съществуват и други инструкции за различни архитектури (пр. NEON инструкции за ARM архитектурите). Инструкциите за 256-битови регистри се наричат Advanced Vector Extensions (AVX), докато инструкциите за 512-битови регистри са известни с общото наименование AVX512.

Една основна функция, която е налична в повечето съвременни централни процесори, е функция за намиране на броя на ненулевите битове в компютърна дума *popcnt*. Тази функция се използва в алгоритмите за намиране на тегло на вектор.

Глава 2

В тази глава са разгледани два основни вида алгоритми, използвани за намирането на тегловен спектър на линеен код - алгоритми от високо ниво, които представят подхода за генериране на нова кодова дума и алгоритми от ниско ниво, които извършват основните изчисления. Алгоритмите от високо ниво са базирани на подход за емулация на вложени цикли, представен в [9].

Алгоритми от високо ниво

Генерирането само на непропорционални кодови думи се извършва чрез прескачане на линейни комбинации, чиито първи ненулев коефициент е различен от 1. За целта се използва помощна матрица, която в ред i съдържа кодова дума, получена като линейна комбинация на i реда на пораждащата матрица.

Основната оптимизация на представения алгоритъм се състои в заменяне на операцията умножение на вектори, която е изчислително тежка, с операция за събиране на вектори. При работа над прости полета това е лесно постижимо, тъй като елементите на просто поле са остатъци по модул p. Операциите събиране и умножение се извършват по модул p. Тогава умножението с елемент от полето може да се замени с добавяне на един и същ ред от пораждащата матрица към текуща линейна комбинация.

Основната идея за оптимизация може да се опише по следния начин: ако текущият ред от пораждащата матрица се добавя към линейна комбинация на i реда с коефициент 1 (добавя се за първи път към някоя линейна комбинация на $i-1$ реда), то той се добавя към ред $(i-1)$ на временната матрица. Така се получава линейна комбинация на i реда. В противен случай (текущият ред на G се добавя с коефициент $\neq 1$ в линейната комбинация), редът се добавя към съществуваща линейна комбинация на i реда. При полетата $\mathbb{F}_2$ и $\mathbb{F}_3$ описанието на алгоритъма може да бъде опростено.

Елементите на съставно поле $\mathbb{F}_q$, където $q = p^m$, могат да се представят като полиноми с коефициенти над крайното поле $\mathbb{F}_p$ и степен по-малка от m, на които се съпоставят вектори. Тогава събирането на вектори над полето е лесно за имплементиране. За оптимизация в представения алгоритъм се използват предварителни изчисления, които се извършват еднократно преди започване на основните изчисления. Предварително се генерира множеството от пораждащи матрици $M = \{G, xG, \ldots, x^{m-1}G\}$. Използва се примитивен пораждащ полином и следователно за примитивния елемент имаме $\alpha = x$. При наличието на множество M, умножението на ред от пораждащата матрица с елемент от полето може да бъде заменено с използването на съответен ред от подходяща матрица от M. За избиране на матрица от M се използва редица на прехода на p-ичен код на Грей.

Алгоритми от ниско ниво

В този раздел са представени подходите за реализиране на алгоритмите

от ниско ниво за събиране на вектори и изчисляване на теглото на вектор. Те са различни в зависимост от стойностите на q. В основата на тези алгоритми е подходящото представяне на елементите на полето за записване в регистър. Разглеждат се две основни представяния за елементите на полето в паметта - побитово и побайтово.

Използването на побитово представяне на елементите и побитови операции може да се разглежда като най-естествения подход за паралелизация. За полетата с 2, 3 и 4 елемента са разработени различни побитови представяния и подходи за извършване на операции [1, 8, 15, 23, 28].

Векторите над поле с два елемента имат естествено двоично представяне - всеки бит може да отговаря за една координата на вектора. Интересен случай е n-мерното векторно пространство, където $n \leq 64$. В този случай повече от половината битове на регистъра не носят информация. За генериране на кодовите думи на код C с дължина $n \leq 64$, може да се използва $[n, k-1]$ подкод C' с пораждаща матрица G', получена от G като е премахнат последния ред g_k. Също така се разглежда съседният клас $g_k + C' = \{g_k + c | c \in C'\}$. Тогава първоначалният код C може да се представи като множеството $C' \cup (g_k + C')$. Чрез подходящо записване на пораждащата матрица и временната матрица се генерират кодови думи от C' и $(g_k + C')$ едновременно. За извършване на операцията за намиране на теглото на вектор, един регистър от 128 бита може да разгледа като масив от две 64-битови компютърни думи. Теглата на две генерирани кодови думи се намират, като се използва *popcnt* инструкция. При $n > 64$ за намиране на теглото се използва същата инструкция, като се сумират получените стойности.

За полета с три елемента съществуват различни подходи за побитово представяне на елементите на полето [1, 8, 15, 23, 28]. Едно такова представяне дава възможност за записване на елемент на полето в два бита, като събирането на елементите се извършва чрез шест побитови операции [15]. Това представяне е използвано в алгоритмите от ниско ниво при полета с три елемента. Изразява се чрез следното изображение $\Pi : \mathbb{F}_3 \to \mathbb{F}_2^2$, където

$$\Pi(0) = (1, 1)$$

$$\Pi(1) = (1, 0)$$

$$\Pi(2) = (0, 1)$$

Изображението Π може да се разшири за вектори над полето до $\pi : \mathbb{F}_3^n \to \mathbb{F}_2^{2n}$, където

$$\pi(v) = (\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n),$$

$$v = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_3^n, \Pi(v_i) = (\alpha_i, \beta_i).$$

Използвайки тези представяния, операцията за събиране на вектори над $\mathbb{F}_3$ може да се реализира чрез 5 операции над регистри. Изображението π също така позволява лесно изчисление на броя на ненулевите елементи на даден вектор. Тъй като елемента 0 се изобразява в $(1, 1)$, за изчислението на теглото на вектор v при използване на изображението

$$\pi(v) = (\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n),$$

е необходимо да се изпълни XOR операция за α и β частите на $\pi(v)$, след което се изпълнява *popcnt* функция.

Елементите на простото поле $\mathbb{F}_p$ могат да бъдат представени като цели числа в интервала $[0, p-1]$. Следователно в случаите, когато $p \neq 2, 3$ може да се използват 8-битови цели числа при $p < 128$. Представеният алгоритъм за събиране на вектори над просто поле използва три инструкции за разширени регистри - събиране, изваждане и съчетаване на регистри. Операцията се извършва за полета с до 64 елемента, защото се използва представяне в паметта чрез знакови типове от данни. За извършване на инструкцията за съчетаване на регистри е от значение дали резултата от изваждането е отрицателни число.

За намирането на теглото на вектор над просто поле са разгледани два основни подхода. Първият подход намира броя на нулевите байтове и използва няколко извиквания на тежка функция *popcnt*. Броя на извикванията зависи от дължината на регистъра. Вторият подход намира броя на нулевите координати като използва маска, побитови операции и едно извикване на *popcnt* функция.

Векторите в $\mathbb{F}_{p^m}^n$ са представени като вектори в $\mathbb{F}_p^{mn}$. Тогава операцията събиране на вектори се свежда до реализацията ѝ над простото поле. Намирането на теглото на вектор отново се свежда до изпълнение на операциите над простото поле, като се използват допълнителни побитови операции. Броят и вида на допълнителните операции зависи от p.

Анализ на ефективността

За да се анализира ефективността на разработените алгоритми са направени сравнения със системите за компютърна алгебра Magma и GAP. Също така са направени сравнения между времето за работа на имплементациите с 128- и 256-битови регистри, както и сравнение между времето за работа на имплементацията с 256-битови регистри и имплементация без векторизация на

алгоритмите. Експерименталните резултати показват, че при използването на побитово представяне ($\mathbb{F}_2$,$\mathbb{F}_4$, $\mathbb{F}_3$,$\mathbb{F}_9$, $\mathbb{F}_{27}$) 256-битовата версия е сравнима със 128-битовата версия. При съставни полета времето за изпълнение с AVX2 е с между 10% и 30% по-малко в сравнение с SSE4.1. В случай на побайтово представяне ($\mathbb{F}_5$, $\mathbb{F}_{25}$), 256-битовата версия е между 1,2 и 1,7 пъти по-бърза от 128-битовата версия.

При простите полета $\mathbb{F}_2$, $\mathbb{F}_3$ и $\mathbb{F}_5$ се наблюдават между 1,4 и 2,5 пъти по-бързи изчисления в сравнение с функцията на Magma, като ускорението се увеличава с увеличаването на дължината на кода. При съставните полета, представената векторизирана версия е между 1,9 ($\mathbb{F}_4, n = 500$) и 43.4 ($\mathbb{F}_{27}, n = 500$) пъти по-бърза в сравнение с Magma. При сравнение с GAP експерименталните резултати представят ускорение между 3,3 пъти ($\mathbb{F}_2$) и 1480 пъти ($\mathbb{F}_{27}$). Тук отново по-големи ускорения се наблюдават при съставните полета.

При сравнение с невекторизирана версия на алгоритъма се наблюдават ускорения между 7,8 ($\mathbb{F}_{25}$) и 58,9 ($\mathbb{F}_2$) пъти. Това се дължи на високия коефициент на векторизация. При побитово представяне, коефициентът на векторизация за 128-битов регистър е между 21 и 128 в зависимост от полето. Тогава, при сравнение с невекторизирана имплементация, използваща таблици за събиране и умножение за операциите с елементи над полето, естествено се получава голямо ускорение.

Векторизация и ефективността на компилаторите

Известно е, че изборът на компилатор също може да повлияе на бързодействието на алгоритмите [2, 6]. За това е направено сравнение между три широко използвани компилатора за езиците C/C++ - gcc (mingw за Windows операционна система), clang и msvc. Експериментални резултати са получени за операционните системи Windows и Ubuntu, като при Ubuntu са разгледани само първите два компилатора. Изчисленията са извършени за двоични полета и 128- и 256-битови регистри. Експерименталните резултати показват, че времената за изпълнение на функция за намиране на тегловен спектър при компилиране с трите избрани компилатора са близки. Наблюдава се, че функциите, компилирани с gcc, са по-подходящи за използване с 256-битови регистри. Функциите, компилирани с clang и msvc, дават по-добро време за изпълнение при 128-битови регистри.

Глава 3

В Глава 3 са описани основни особености при векторизиране на алгоритми с разширените инструкции AVX512 и Neon. Семейството от инструкции AVX512 е разделено на различни подкатегории. Някои от особеностите на AVX512 инструкциите, които са налични в някои съвременни архитектури от вида x86, са следните:

- Маскиращи регистри (opmask) - използват се за ефективно сливане и условно записване в резултатните регистри.
- *AVX512Foundation* - основни операции за работа с регистрите (аритметични операции, логически операции, побитови операции, функции за записване на данни в паметта и др.).
- *AVX512VPOPCNTDQ* - функции, позволяващи намирането на теглото на целочислени елементи, записани в регистър. Функциите от тази подкатегория са основни за ефективността на разработените алгоритми.
- *AVX512BW* (*Byte and Word Instructions*) - разширява основните функционалности, дефинирани в *AVX512Foundation*, като добавя функции за 8 и 16-битови цели числа. Такива функции се използват за имплементация на алгоритмите за събиране на вектори над прости полета с побайтово представяне.

Някои от основните характеристики на NEON инструкциите за ARM архитектури са следните:

- Векторизация чрез записване на еднотипни елементи в 64- или 128-битови регистри.
- Различен синтаксис, който описва типа на данните, които ще бъдат записани в регистъра, неговата големина и броя на елементите, които ще бъдат записани в регистъра. Например int8x16_t съдържа 16 целочислени елемента с големина 8 бита.
- Наличие на всички операции за сравнение ($<, >, =, \leq, \geq$)
- Функциите за намиране на броя на ненулеви битове са разработени като намират броя им за всеки байт, записан в регистър. За намирането на теглото на цял векторен регистър е необходимо да се съберат резултатните стойности, което е възможно чрез една функция.

Беззнакови типове от данни и операции със сатурация

Разширените векторни инструкции за x86 и ARM архитектури разполагат с функции със сатурация. При изпълняване на някоя функция, използвайки сатурация, резултатът от операцията е равен на резултата от стандартното изпълнение на операцията, когато не се излиза от диапазона на дадения тип от данни. Ако резултатът от операцията е извън диапазона на типа от данни, записан в регистъра, за някой елемент, то в изходния регистър за съответния елемент се записва променлива, съдържаща само нулеви битове, когато стойността е по-малка от долната граница за валидните стойности на дадения тип от данни. Когато резултатът е по-голям от горната граница на валидните стойности, в регистъра на съответната позиция се записва променлива с подходяща дължина, съдържаща само ненулеви битове.

Функциите, реализирани чрез сатурация, позволяват алгоритмите за събиране на вектори да бъдат реализирани за вектори над крайни прости полета с $q < 128$ елемента и с голям фактор на векторизация. Това позволява да се извършат изчисления за по-големи полета, като се използва същото представяне и ресурс, използвани за реализирането на алгоритмите от ниско ниво. За целта елементите на полето се записват в беззнакови типове от данни.

При реализиране на операцията събиране на вектори чрез беззнакови типове от данни и функции със сатурация се използват 7 леки операции (аритметични, операции за сравнение и побитови операции) за инструкциите SSE, AVX и NEON. При имплементация с AVX512 операцията се реализира чрез 4 инструкции, благодарение на маскиращите регистри. При реализация чрез AVX512, функцията за намиране на тегло на вектор се изпълнява чрез 2 функции - сравнение и *popcnt* инструкция за резултатния 64-битов регистър.

Експериментални резултати

Анализирана е ефективността на инструкциите AVX512 и NEON, като е използван алгоритъма за намирането на спектър на линеен код, реализиран чрез беззнакови типове от данни и инструкции със сатурация. Времето за работа чрез AVX512 е между 1,5 и 4,5 пъти по-бързо в сравнение с SSE инструкции. Времето за работа чрез NEON е между 1,3 и 2,9 пъти по-бързо в сравнение с SSE инструкции. При имплементация на алгоритмите от ниско ниво чрез побитово представяне и $n \leq 500$, реализациите с AVX512 и NEON инструкциите имат близки времена за изпълнение. Експерименталните резултати също по-

казват, че при коефициент на използване, по-малък от 1/2, е подходящо да се използват регистри с дължина 128 бита.

Глава 4

Двоичен линеен $[n, k]$ код C се нарича самодопълнителен, ако за всяка кодова дума $x \in C$, нейното допълнение $\overline{x}$, където $x + \overline{x} = \mathbf{1}$, също е кодова дума ($\overline{x} \in C$). За обема на двоичен линеен $[n, k, d]$ самодопълнителен код е в сила границата на Грей-Ранкин, дадена чрез следното неравенство, ако дясната страна е положително число:

$$|C| \leq \frac{8d(n-d)}{n-(n-2d)^2}. \tag{1}$$

Тази граница е в сила за линейни и нелинейни двоични самодопълнителни кодове. Самодопълнителните кодове, които са разгледани в тази работа са двоични линейни кодове. Параметрите на линейни кодове, които достигат равенство при границата на Грей-Ранкин, са

$$\begin{gathered}[2^{2m-1}-2^{m-1}, 2m+1, 2^{2m-2}-2^{m-1}], \\ [2^{2m-1}+2^{m-1}, 2m+1, 2^{2m-2}].\end{gathered} \tag{2}$$

Самодопълнителни кодове, които достигат границата на Грей-Ранкин са широко изучавани. От особен интерес са техните подкодове, поради връзките им със силно регулярни графи [10, 11, 16], квазисиметрични SDP дизайни [22, 27], бент [14, 17] и векторни бент функции [19], четни двоични кодове и други. Броят на нееквивалентните кодове е равен на броя на неизоморфните SDP дизайни. Доказано е, че техният брой нараства експоненциално с нарастване на размерността [27].

Проективнодопълнителни и самодопълнително еквивалентни кодове

Дефиниция 1. *Нека C е проективен линеен $[n, k]$ код с пораждаща матрица G. Разглеждаме матрицата $S'_k = (G|\overline{G})$, която е пораждаща матрица на код, еквивалентен на симплекс кода с размерност k. Кодът $\overline{C}$ с пораждаща матрица $\overline{G}$ наричаме проективно допълнителен код на C.*

Разгледани са следните две конструкции:

- Нека C е $[n, k-1]$ линеен код, който не е самодопълнителен и $\widehat{C} = C \cup (\mathbf{1} + C)$. Ако G е пораждаща матрица на C, то матрицата $\widehat{G} = \begin{pmatrix} 1 & \dots & 1 \\ & G & \end{pmatrix}$ е пораждаща матрица на $\widehat{C}$.
- Нека C е $[n-1, k-1]$ линеен код, който не е самодопълнителен и $\widehat{C'} = (0|C) \cup (1|\mathbf{1} + C)$. Ако G е пораждаща матрица на C, то матрицата $\widehat{G'} = \begin{pmatrix} 1 & \dots & 1 & \\ & G & & 0 \end{pmatrix}$ е пораждаща матрица на $\widehat{C'}$.

Чрез така дефинираните кодове $\widehat{C}$ и $\widehat{C'}$ се дефинира следната релация на еквивалентност:

Дефиниция 2 (Самодопълнително еквивалентни кодове)**.** *Нека разгледаме двоични линейни кодове, които не съдържат вектора* ***1****.*

- *Ако C_1 и C_2 са $[n, k-1]$ кодове, то те се наричат самодопълнително еквивалентни, ако кодовете $\widehat{C_1}$ и $\widehat{C_2}$ са еквивалентни самодопълнителни кодове с дължина n.*
- *Ако C_1 и C_2 са съответно с параметри $[n-1, k-1]$ и $[n, k-1]$, то те са самодопълнително еквивалентни, ако съответните им кодове $\widehat{C_1'}$ и $\widehat{C_2}$ са еквивалентни самодопълнителни кодове с дължина n. Аналогично, ако C_1 и C_2 са съответно с параметри $[n, k-1]$ и $[n-1, k-1]$, то те са самодопълнително еквивалентни, ако съответните им самодопълнителни кодове с дължина n са еквивалентни.*
- *Ако C_1 и C_2 са $[n-1, k-1]$ кодове, то те са самодопълнително еквивалентни, ако кодовете $\widehat{C_1'}$ и $\widehat{C_2'}$ са еквивалентни самодопълнителни кодове с дължина n.*

Разглеждайки тази релация на еквивалентност, за самодопълнителен код C, всички подкодове, които не съдържат вектора $\mathbf{1}$, са в един клас на самодопълнителна еквивалентност.

Фамилии от кодове с две и три тегла

За код достигащ границата на Грей-Ранкин с размерност $k+1$ и следните означения ($k = 2m$):

$$t_k = 2^{k-2}, t_{k^\pm} = t_k \pm 2^{m-1},$$
$$T_{k^\pm} = 2^{k-1} \pm 2^{m-1}$$

дефинираме:

- фамилии от кодове с две тегла:
  - Φ_{k^-}: $[T_{k^-}, k, \{t_{k^-}, t_k\}]$
  - Φ_{k^+}: $[T_{k^+}, k, \{t_k; t_{k^+}\}]$
  - Φ'_{k^-}: $[T_{k^-} - 1, k, \{t_{k^-}, t_k\}]$
  - Φ'_{k^+}: $[T_{k^+} - 1, k, \{t_k; t_{k^+}\}]$
- фамилии кодове с три тегла:
  - Ψ_k: $[2^k, k+1, \{T_{k^-}; 2^{k-1}; T_{k^+}\}]$
  - Ψ'_k: $[2^k - 1, k+1, \{T_{k^-}; 2^{k-1}; T_{k^+}\}]$

Представени са следните връзките между така дефинираните фамилии от кодове:

- Проективнодопълнителни двойки кодове: Φ_{k^+} и Φ'_{k^-}; Φ_{k^-} и Φ'_{k^+}; Ψ_k и Ψ'_k
- От код в едно от семействата Φ_{k^-}, Φ_{k^+}, Φ'_{k^-}, Φ'_{k^+} може да се конструират кодове от другите три семейства.
- От код от семейството Ψ_k може да се конструира код в Ψ'_k и обратно.
- Ако C е код от фамилията $\Phi_{k^\pm}$ (или $\Phi'_{k^\pm}$), тогава остатъчният му код по кодова дума с тегло $t_{k^\pm}$ принадлежи на фамилията Ψ_k (съотв. Ψ'_k).
- Нека с Ψ_{S_k} и Ψ'_{S_k} означим кодове от съответните фамилии Ψ_k и Ψ'_k, които съдържат симплексния код като подкод. Ако $C \in \Psi_{S_k}$ (съотв. $C \in \Psi'_{S_k}$) и $v \in C$ с тегло $T_{k^\pm}$ тогава $Res(C, v) \in \Phi_{k^\mp}$ (съотв. $\Phi'_{k^\mp}$).

Конструкции за построяване на кодове от семействата Φ_{k+2} чрез кодове с размерност k

За кодовете от Φ_k може да се разгледат конструкции, които позволяват построяването на кодове от размерност $k+2$ като се използват кодове с размерност k. Чрез тази конструкция може да се дефинира безкрайна фамилия от кодове.

Теорема 1. *Нека $A \in \Phi_{k^\pm}$ е код с пораждаща матрица G_A и B е код, породен*

от матрицата

$$\left(\begin{array}{c|c|c|c} G_A & G_A & G_A & \overline{G_A} \\ 1\ldots1 & 1\ldots1 & 0\ldots0 & 0\ldots0 \\ 0\ldots0 & 1\ldots1 & 1\ldots1 & 0\ldots0 \end{array}\right).$$

Тогава $B \in \Phi'_{(k+2)^{\pm}}$.

Теорема 2. *Нека* $A \in \Phi'_{k^{\pm}}$ *има пораждаща матрица* G_A *и* B *има пораждаща матрица*

$$\left(\begin{array}{c|c|c|c} G_A\ 0 & G_A\ 0 & G_A\ 0 & \overline{G_A} \\ 1\ldots1 & 1\ldots1 & 0\ldots0 & 0\ldots0 \\ 0\ldots0 & 1\ldots1 & 1\ldots1 & 0\ldots0 \end{array}\right).$$

Тогава $B \in \Phi_{(k+2)^{\pm}}$.

Теорема 3. *От всеки код* $C \in \Phi_k$ *могат да бъдат конструирани кодове в* Φ_{k+2l} *за всички цели числа* $l \geq 1$.

Изчислителни резултати

Изчислителните резултати са получени чрез модула *Generation* на програмата *QextNewEdition*[7] и разработената библиотеката. Следва обобщение на получените изчислителни резултати:

- Конструирани са 322039 нееквивалентни $[63, 7; \{28, 32, 36\}]$ кода от 7 нееквивалентни $[35, 6; \{16, 20\}]$ кода.
- Конструирани са 91337 нееквивалентни $[119, 8; \{56, 64\}]$ кодове от 4 нееквиваленти $[63, 7; \{28, 32, 36\}]$ кода.
- Конструирани са 2946 нееквивалентни самодопълнителни кода с параметри $[120, 9; \{56, 64, 120\}]$ от $[119, 8; \{56, 64\}]$ кодове, като от тях получаваме:
  - 175213 подкода с дължина 120 и размерност 8;
  - 156763 подкода с дължина 119 и размерност 8.
- При използване на SC-lifting конструкцията върху 80 нееквивалентни кода с параметри $[119, 8; \{56, 64\}]$, са получени:
  - 80 нееквивалентни $[496, 10; \{240, 256\}]$ кода;
  - 80 нееквивалентни $[2015, 12; \{992, 1024\}$ кода.

Глава 5

Разработените алгоритми са включени в библиотека за намиране на тегловни инварианти на линейни кодове **LinCodeWeightInv** над поле $\mathbb{F}_q$, където $q \leq 64$. Тези алгоритми са реализирани, като са използвани различни набори от инструкции и могат да бъдат изпълнени на централни процесори с ARM и x86 архитектури. Библиотеката включва следните шест основни функции:

- Изчисление на тегловен спектър на линеен код.
- Намиране на минималното разстояние на линеен код.
- Намиране на броя на кодовите думи с дадено тегло w.
- Намиране на броя на кодовите думи с тегло по-малко от w.
- Определяне дали съществуват кодови думи с дадено тегло w.
- Определяне дали съществуват кодови думи с тегло по-малко от дадено w.

Основните данни, необходими за изчисленията, са параметрите n, k и q и пораждаща матрица на кода. За всяка от интерфейсните функции са създадени следните подходи за въвеждане на данните:

- Четене на входните данни от файл. Резултатите се записват във изходен файл с подходящо наименование спрямо интерфейсната функция.
- Генериране на псевдослучаен код по зададени стойности на k, n, q. Функцията приема като параметри стойности за k, n, q, като за тях се генерира пораждаща матрица.
- Изчисление за записана пораждаща матрица в паметта като двумерен масив. Функцията получава като параметри динамичен двумерен масив от тип *int*, параметрите на кода n, k, q и булева променлива, която показва дали елементите на полето в пораждащата матрица са записани чрез адитивен или мултипликативен запис (променливата има стойност *true*, ако е използван мултипликативен запис).

Основните функционалности на разработената библиотека се базират на намирането на тегловния спектър на кода. За работата на основните функции са разработени над 36 имплементации на базови функции, необходими за извършване на изчисленията. За работа с библиотеката са разработени два

допълнителни модула - интерфейсна програма, която има за цел да улесни използването на библиотеката и модул за тестване и верификация. Този модул позволява тестване на всички функционалности за коректност на изчисленията и времето за изпълнението им. За целта се използва предварително подготвен входен файл съдържащ линейни кодове за всички полета $\mathbb{F}_q$, $q \leq 64$ и съответните тегловни спектри във вида $\{A_i\}$, където $i = 0, \ldots, n$, генериран с различен софтуер. В изходен файл *Results* се записва средно време за изпълнение на изчисленията за всяко поле при получаване на правилен резултат.

За създаването на преносим софтуер и подробно ръководство за ползване са използвани системите CMake и Doxygen. CMake е продукт, който позволява създаване на проект на C/C++. Чрез този софтуер се контролират основни параметри за компилирането на библиотеката. Създадено е подробно ръководство за използването на библиотеката, което включва описание на основните ѝ компоненти и начините за компилиране, инсталиране и тестване. За целта е използвана системата Doxygen, която създава подробно описание (в нашия случай над 50 страници), като се използват стандартизирани коментари в програмния код. В тази глава са описани основни характеристики на тези системи, които са използвани при създаването на библиотеката. Описана е цялостната структура на библиотеката, начините за компилиране и инсталиране като се използва CMake.

Научно-приложни приноси

- Разработване на алгоритъм за генериране на непропорционални кодови думи, който работи за съставни полета само чрез събиране на два вектора.
- Разработване на алгоритъм за събиране на вектори чрез SSE, AVX и AVX512 инструкции при използване на побитово представяне на елементите на полета $\mathbb{F}_2$, $\mathbb{F}_4$ и полетата с характеристика 3.
- Разработване на алгоритъм за събиране на вектори чрез SSE, AVX и AVX512 инструкции инструкции при байтово представяне за прости и съставни полета с до 64 елемента, включително, и намиране на теглото на вектор чрез едно извикване на *popcnt* инструкция.
- Анализ на ефективността на работата на различни компилатори с SSE и AVX инструкции при векторизация.
- Изучаване на характеристиките на наборите от инструкции AVX512 и NEON и анализиране на тяхната ефективност.

- Анализиране на ефективността на различните видове инструкции в архитектури от вида x86.
- Разработване на алгоритъм за събиране на вектори чрез SSE, AVX и AVX512 при представяне на елементите на полета с до 128 елемента чрез беззнакови цели числа.
- Дефиниране на фамилии от кодове с две и три тегла, свързани с кодове, достигащи границата на Грей-Ранкин, и описание на връзките между отделните фамилии.
- Разработване на конструкция за построяване на кодове от дадена фамилия с размерност $k + 2$ чрез кодове от съответна фамилия с размерност k.
- Разработване на математически софтуер (библиотека LinCodeWeightInv) за намиране на теглови инварианти на линейни кодове над полета с до 64 елемента, включително, като се използват SSE4.1, AVX2 и AVX512 инструкции за x86 архитектури и NEON инструкции за ARM архитектури.
- Представяне на особености и основните акценти при създаването на математически софтуер с отворен код.

Апробация на резултатите

Резултатите, включени в дисертацията, са получени самостоятелно [P2, P3] и в съавторство с:

- Буюклиев [P1, P5, P6]
- Буюклиев и Буюклиева [P4]

Включените статиите са публикувани или изпратени за рецензия в:

- *Science Series-Innovative STEM Education* [P1, P2]
- *2022 International Conference Automatics and Informatics (ICAI)* [P3]
- *Mathematics* [P4]
- *International Conference on Large-Scale Scientific Computing* [P5]
- *ACM Transactions on Mathematical Software* [P6]

Резултати по дисертацията са докладвани на:

- Конференция с международно участие *"Иновативно STEM образование"* 2021-2022 г. [D1, D3]
- Национален семинар по теория на кодирането *Проф. Стефан Додунеков*, 2021-2022 г. [D2, D5]
- International Conference Automatics and Informatics, Varna, Bulgaria, 2022 г.[D4]
- 4-th Interdisciplinary PhD Forum with International Participation, Sandanski, Bulgaria, 2023 г. [D6]
- 14th International Conference on Large-Scale Scientific Computations, Sozopol, Bulgaria, 2023 г. [D7]
- Семинар *"HPC for Mathematics and Applications"*, София, България, 2023 г. [D8]
- International Conference *"Cryptography and Coding Theory"*, Perugia, Italy, 2023 г. [D9]

Благодарности

Благодаря на всички съавтори за насоките и чудесния работен процес. Благодаря на колегите от секция "Математически основи на информатиката"към Института по математика и информатика (ИМИ) на Българската академия на науките (БАН) за създадената работна атмосфера, съветите и конструктивната критика. Получената обратна връзка през годините несъмнено допринесе за цялостната стойност на изследванията. Също така бих искала да изразя своята признателност към ръководството и служителите на ИМИ-БАН за предоставените възможности, подкрепа и достъп до изчислителните ресурси. Благодаря и на колегите от Факултета по математика и информатика към Великотърновския университет за възможността да се запозная и с друг начин за усвояване на знания - чрез преподаване.

Специални благодарности отправям към моя научен ръководител проф. д.м.н. Илия Буюклиев за търпението, подкрепата, конструктивните съвети и интересните теми, с които ме запозна при работата върху дисертационния труд. Съветите и придобитите умения ще ми помагат и за напред в развитието ми както в академично отношение, така и като личност.

Накрая бих искала да изразя признателността си към моя съпруг за търпението и подкрепата, които оказа през академичния ми път и във всяко едно отношение. Също така му благодаря, че е мой неотлъчен спътник във всички начинания.

Библиография

[1] Байчева, Ц. и Манев, Кр. Намиране на линейната обвивка на множество от вектори над крайно поле с характеристика различна от 2. *Доклади на XXIII Пролетна конференция на СМБ* (1994), 313–318.

[2] Amiri, H., and Shahbahrami, A. Simd programming using intel vector extensions. *Journal of Parallel and Distributed Computing 135* (2020), 83–100.

[3] Berlekamp, E., McEliece, R., and Van Tilborg, H. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory 24*, 3 (1978), 384–386.

[4] Bikov, D., and Bouyukliev, I. Parallel fast walsh transform algorithm and its implementation with cuda on gpus. *CYBERNETICS AND INFORMATION TECHNOLOGIES 18*, 5 (2018), 21–43.

[5] Bikov, D., Bouyukliev, I., and Dzhumalieva-Stoeva, M. Boolsplg: A library with parallel algorithms for boolean functions and s-boxes for gpu. *Mathematics 11*, 8 (2023), 1864.

[6] Botor, T., and Habiballa, H. Compiler optimization for scientific computation in c/c++. In *International Conference of Computational Methods in Sciences and Engineering 2018 (ICCMSE 2018)* (2018), vol. 2040, p. 030004.

[7] Bouyukliev, I. The program generation in the software package qextnewedition. In *International Congress on Mathematical Software* (2020), Springer, pp. 181–189.

[8] Bouyukliev, I., and Bakoev, V. Efficient computing of some vector operations over gf(3) and gf(4). *Serdica Journal of Computing 2*, 2 (2008), 137–144.

[9] Bouyukliev, I., and Bakoev, V. A method for efficiently computing the number of codewords of fixed weights in linear codes. *Discrete applied mathematics 156*, 15 (2008), 2986–3004.

[10] Bouyukliev, I., Fack, V., Willems, W., and Winne, J. Projective two-weight codes with small parameters and their corresponding graphs. *Designes, Codes and Cryptography 41* (2006), 59–78.

[11] Calderbank, R., and Kantor, W. The geometry of two-weight codes. *Bulletin of the London Mathematica Society 18*, 2 (1986), 97–122.

[12] Carlet, C., Charpin, P., and Zinoviev, V. Codes, bent functions and permutations suitable for des-like cryptosystems. *Designs, Codes and Cryptography 15* (1998), 125–156.

[13] Carlet, C., Crama, Y., and Hammer, P. L. Boolean functions for cryptography and error-correcting codes., 2010.

[14] Carlet, C., and Mesnager, S. Four decades of research on bent functions. *Designs, codes and cryptography 78*, 1 (2016), 5–50.

[15] Coolsaet, K. Fast vector arithmetic over f_3. *Bulletin of the Belgian Mathematical Society-Simon Stevin 20*, 2 (2013), 329–344.

[16] Delsarte, P. Weights of linear codes and strongly regular normed spaces. *Discrete Mathematics 3*, 1-3 (1972), 47–64.

[17] Dillon, J., and Schatz, J. Block designs with the symmetric difference property. In *Proceedings of the NSA Mathematical Sciences Meetings* (1987), Fort Meade, USA, The United States Government, pp. 159–164.

[18] Dimitrov, M., and Esslinger, B. Cuda tutorial–cryptanalysis of classical ciphers using modern gpus and cuda. *arXiv preprint arXiv:2103.13937* (2021).

[19] Ding, C., Munemasa, A., and Tonchev, V. D. Bent vectorial functions, codes and designs. *IEEE Transactions on Information Theory 65*, 11 (2019), 7533–7541.

[20] Flynn, M. J., and Rudd, K. W. Parallel architectures. *ACM computing surveys (CSUR) 28*, 1 (1996), 67–70.

[21] Fog, A. Optimizing software in c++: An optimization guide for windows, linux, and mac platforms, 2004.

[22] Gulliver, T., and Harada, M. Codes of lengths 120 and 136 meeting the grey-rankin bound and quasi-symmetric designs. *IEEE Transactions on Information Theory 45*, 2 (1999), 703–706.

[23] Harrison, K., Page, D., and Smart, N. P. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *LMS Journal of Computation and Mathematics 5* (2002), 181–193.

[24] Hu, L., Che, X., and Zheng, S.-Q. A closer look at gpgpu. *ACM Computing Surveys (CSUR) 48*, 4 (2016), 1–20.

[25] Huffman, W. C., and Pless, V. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge, UK, 2003.

[26] Joux, A. *Algorithmic cryptanalysis*. Chapman and Hall/CRC, 2009.

[27] Jungnickel, D., and Tonchev, V. D. Exponential number of quasi-symmetric sdp designs and codes meeting the grey-rankin bound. *Designs, Codes and Cryptography 1*, 3 (1991), 247–253.

[28] Kawahara, Y., Aoki, K., and Takagi, T. Faster implementation of ηtpairing over gf(3m) using minimum number of logical instructions for gf(3)-addition. In *Pairing-Based Cryptography – Pairing 2008* (Berlin, Heidelberg, 2008), S. D. Galbraith and K. G. Paterson, Eds., Springer Berlin Heidelberg, pp. 282–296.

[29] Kirk, D. B., and Wen-Mei, W. H. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[30] MacWilliams, F., and Sloane, N. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 1977.

[31] Mattson, T., He, Y., and Koniges, A. *The OpenMP Common Core*. The MIT Press, 2019.

[32] Mullen, G. L., and Panario, D. *Handbook of Finite Fields*. Chapman and Hall, 2013.

[33] Padua, D. *Encyclopedia of parallel computing*. Springer Science & Business Media, 2011.

[34] Quinn, M. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Inc., 2004.

[35] Topalova, S., and Zhelezova, S. Parallelisms of pg (3, 4) with a great number of regular spreads. In *International Conference on Large-Scale Scientific Computing* (2023), Springer, pp. 444–452.

[36] Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H., Huff, K. D., Mitchell, I. M., Plumbley, M. D., et al. Best practices for scientific computing. *PLoS biology 12*, 1 (2014), e1001745.

Изнесени доклади

[D1] Pashinska, M., Bouyukliev, I.; Using AVX instructions for optimization of linear binary code weighting algorithms; 3rd National Scientific Conference with International Participation "Innovative STEM Education"; Veliko Tarnovo, Bulgaria, 25.04.2021 - 27.04.2021

[D2] Pashinska-Gadzheva, M., Bouyukliev, I.; SSE 4.1 optimizations for algorithm for calculating the Weight Spectrum of linear codes, National Coding Theory workshop "Professor Stefan Dodunekov"; Zlatograd, Bulgaria; 04.11.2021 - 07.11.2021

[D3] Pashinska-Gadzheva, M.; Build systems for generating independent software; Fourth International Conference "Innovative STEM Education"; Veliko Tarnovo, Bulgaria; 16.03.2022 - 19.03.2022

[D4] Pashinska-Gadzheva, M.; Comparison of compiler efficiency with SSE and AVX instructions; International Conference Automatics and Informatics (ICAI) 2022; Varna, Bulgaria; 06.10.2022 - 08.10.2022

[D5] Pashinska-Gadzheva, M., Bouyukliev, I., Bouyuklieva, S.; Two-weight codes and Grey-Rankin bound, National Coding Theory workshop "Professor Stefan Dodunekov"; Arbanasi, Bulgaria; 09.11.2022 - 13.11.2022

[D6] Pashinska-Gadzheva, M.; Optimization and Parallelization of Algorithms Connected to Coding Theory, 4-th Interdisciplinary PhD Forum with International Participation; Sandanski, Bulgaria; 16.05.2023 - 19.05.2023

[D7] Pashinska-Gadzheva, M., Bouyukliev, I.; About methods of vector addition over finite fields using extended vector register; 14th International Conference

on Large-Scale Scientific Computations; Sozopol, Bulgaria; 05.06.2023 - 09.06.2023

[D8] Pashinska-Gadzheva, M., Bouyukliev, I.; Library for Computing Weight Invariants of Linear Codes Using Vectorization; HPC for Mathematics and Applications, Sofia, Bulgaria; 28.06.2023 - 28.06.2023

[D9] Pashinska-Gadzheva, M., Bouyukliev, I.; About Weight Invariants of Linear Codes and Vectorization with SSE and AVX Instruction Sets, Cryptography and Coding Theory; Perugia, Italy; 21.09.2023 - 22.09.2023

Публикации

[P1] Pashinska M., Bouyukliev I., Utilizing AVX Instruction Set for Optimizing Algorithms for Weight Characteristics of Binary Linear Code. Science Series "Innovative STEM Education"IMI-BAS, 2021, ISSN:2683-1333, 151-156

[P2] Pashinska-Gadzheva, M., Build Systems for Generating Independent Software. Science Series "Innovative STEM Education 4, IMI-BAS, 2022, ISSN:2683-1333, 62-68

[P3] Pashinska-Gadzheva, M. Comparison of compiler efficiency with SSE and AVX instructions. International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2022, pp. 56-59, doi: 10.1109/ICAI55857.2022.9960080

[P4] Bouyukliev, I.; Bouyuklieva, S.; Pashinska-Gadzheva, M. On Some Families of Codes Related to the Even Linear Codes Meeting the Grey–Rankin Bound. Mathematics 2022, 10, 4588, **IF:2.2 SJR: 0.592 Q1**, https://doi.org/10.3390/math10234588

[P5] Pashinska-Gadzheva, M., Bouyukliev, I. About Methods of Vector Addition over Finite Fields Using Extended Vector Registers. In: Lirkov, I., Margenov, S. (eds) Large-Scale Scientific Computations. LSSC 2023. Lecture Notes in Computer Science, Springer, Cham. 2024, vol 13952. pp 427–434, **SJR: 0.606 (2023)**, https://doi.org/10.1007/978-3-031-56208-2_44

[P6] Pashinska-Gadzheva, M., Bouyukliev, I., LinCodeWeightInv: Library for Computing theWeight Distribution of Linear Codes Over Finite Fields,

submited for review after second revision to ACM Transactions on Mathematical Software

Списък на цитирания

[P3] Pashinska-Gadzheva, M. Comparison of compiler efficiency with SSE and AVX instructions. International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2022, pp. 56-59, doi: 10.1109/ICAI55857.2022.9960080

1. Błażejowski, M. Which C compiler and BLAS/LAPACK library should I use: gretl's numerical efficiency in different configurations. *Computational Statistics*, (2024), 1-26.

2. Izrailov, K. GREMC: Genetic Reverse-Engineering of Machine Code to Search Vulnerabilities in Software for Industry 4.0. Predicting the Size of the Decompiling Source Code. *2024 International Russian Smart Industry Conference*, IEEE, 2024, 622-628.