**ЮГОЗАПАДЕН УНИВЕРСИТЕТ „НЕОФИТ РИЛСКИ”**
**Природо-математически факултет**
**Катедра ИНФОРМАТИКА**
*2700 Благоевград, ул. Иван Михайлов 66;*
*073 / 88 55 01; info@swu.bg ; www.swu.bg*

# А В Т О Р Е Ф Е Р А Т

# CAS (СКА) - Нови стратегии и техники

***Костас Зотос***

На дисертация за придобиване на образователна и научна степен „Доктор” в:
Област на висшето образование: 4. Природни науки, математика и информатика,
Професионално направление: 4.6. Информатика и компютърни науки
Докторска програма: Информатика

Научен ръководител: доц. д-р Ирена Атанасова

Благоевград, 2024г.

*Дисертационният труд е обсъден и предложен за защита на заседание на катедра „Информатика” на Природо-математическия факултет на ЮЗУ „Неофит Рилски “ - Благоевград.*

*Съдържанието на дисертацията включва увод, шест глави, заключения, използвана литература и приложения. Текстът е в обем от 81 страници, включва 18 таблици и 18 фигури. Цитираната литература обхваща 67 заглавия на английски език.*

*Защитата на дисертационния труд ще се проведе на 07.03.2025 г. от 12,00 ч., УК1, зала 1432 на ЮЗУ „Неофит Рилски “ – Благоевград.*

*Материалите за защитата са на разположение в канцеларията на катедра „Информатика” на ЮЗУ „Неофит Рилски ” – Благоевград – стая 1461.*

**Резюме**

Системите за компютърна алгебра – СКА (СКА) са софтуер, който използва символни изчисления за извършване на изчисления. С тях математическите изрази могат лесно да бъдат начертани и решени в аналитичен формат. Почти всички науки използват тези системи и учените разчитат в голяма степен на тях. В училищата и университетите те са едни от най-основните инструменти. Въпреки че тези системи са мултиинструменти и изпълняват почти всеки математически проблем с голяма точност, трябва да ги управляваме по определен начин, като предварително имаме предвид някои основни техники.

Тази дисертация има за цел да проучи дали някои техники и стратегии подобряват ефективността на СКА. Изследването се фокусира предимно върху най-известните СКА и се основава главно на проучването на ръководствата за СКА и валидни сравнения, направени от научната общност. Документациите бяха основните източници на данни. За да можем да направим някои полезни изводи, беше направено сравнение на техните характеристики и ефективност при конкретни проблеми. Резултатите показват множество начини, които можем да използваме за подобряване на производителността.

**Ключови думи:** *Системи за компютърна алгебра* (*СКА); СКА оптимизация; СКА производителност; MATLAB; Mathematica;*

## Глава 1

### 1.1 Въведение

Предизвикателство е да се сравняват СКА. Малко изследвания в литературата са се осмелили да ги сравнят. Това се дължи на техните философски позиции и области на специализация в рамките на математиката. Освен това изискванията, функциите и ограниченията на различни хардуерни системи, операционни системи и уеб браузъри могат да варират и да повлияят на функционалността на СКА. Прекомерните изисквания към компютърната памет и процесорната мощност могат да възникнат от широкомащабни числови изчисления. Някои анализатори на СКА смятат, че тъй като компютърният хардуер напредва толкова бързо, няма нужда да се оптимизира производителността на СКА. Изследователите и софтуерните инженери обаче обикновено имат по-малка толерантност към софтуерния дизайн, който не е подходящ. Общата идея на техническите пазари е, че печалбите в производителността трябва да нарастват постоянно.

**Целта на тази дисертация** е да провери дали има техники, които потребителят на СКА може да приложи за подобряване на времето за работа и производителността на системата. В допълнение, целта е да се проучи и анализира еволюцията на тези системи и в комбинация с нововъзникващите технологии да се намерят начини да се направи СКА по-ефективен и удобен

за потребителя. За постигането на тази цел е необходимо изследването на тези системи чрез ръководства за потребителя и валидни сравнения, направени от научната общност. Следователно изследването на потенциала на всяка и тяхното историческо развитие е първият етап от работата. След това трябва да се изберат подходящите техники и стратегии въз основа на времето за изпълнение, необходимите ресурси и ефективността според проведените научни изследвания и спецификациите на всеки СКА. След това важна задача е да се намери коя СКА е за предпочитане и за какъв тип математически проблем. И накрая, трябва да се търсят нови начини за оптимизация чрез изкуствения интелект и други нововъзникващи технологии. И така, изследователските въпроси, на които ще се отговори в тази дисертация, са следните: *а)* Има ли настройки в СКА, които могат да ни дадат по-добра производителност? *б)* Има ли нови технологии и техники, които могат да бъдат използвани? *в)* Какви трябва да бъдат бъдещите СКА и къде трябва да се фокусират?

Тази дипломна работа се състои от шест глави. Втора глава прави преглед на литературата, за да се намерият подобни научни публикации, занимаващи се с СКА оптимизация. В глава 3 се разглеждат нови техники и технологии. В 4-та глава се анализират различни техники за оптимизация на СКА. В 5-та глава резултатите от изследването са описани подробно. И накрая, глава 6 представя публикуваните статии на автора, свързани с тази дисертация и приносите на дисертацията.

### 1.2 Системи за компютърна алгебра

**Системите за компютърна алгебра (СКА)** манипулират математически изрази по начин, подобен на начина, по който един учен би извършил изчисления на хартия. Има забележителни разлики в специфични приложения и характеристики на тези системи [1]. В зависимост от това как се използват, СКА могат да бъдат широко категоризирани в две групи: *а)* СКА с общо предназначение или системи, които включват функции за повечето математически области (напр. *MATLAB, Maple и Mathematica*) и б*)* специални-целеви СКА, които се фокусират върху определени математически области (напр. *CoCoA, DELiA, SINGULAR* и т.н.).

### 1.3 Основните компоненти на СКА

Основните части на системите за компютърна алгебра са както следва:

- **Потребителски интерфейс** (графичен и команден интерфейс).
- **Интерпретатор,** опростител **и** език **за програмиране.**
- **Библиотека**.
- **Вътрешна аритметика,** способна да манипулира математически обекти символно, графично и числено (напр. произволна точност).
- **Мениджър на паметта.**
- **Графичен редактор** за създаване на визуални диаграми.

Дали символната СКА *съхранява*, *манипулира* и *изпълнява* данни на един и същ език е друг решаващ компонент. Някои СКА(като *Reduce*, *Axiom* и *Sympy*) използват един и същ език, докато други (като *Maple* и ) имат различен език за изпълнение в основата. И накрая, някои СКА (като

**Глава 2**

**2.1 Литературен преглед**

Системите за компютърна алгебра са област от интерес за учените поради голямото им значение. Трудно е да ги сравняваме. Малко изследователи в литературата са направили това. Това се дължи на техните философски позиции и области на специализация в рамките на математиката. Освен това изискванията, функциите и ограниченията на различни хардуерни и операционни системи могат да варират и да повлияят на функционалността на СКА. Прекомерните изисквания към компютърната памет, консумацията на енергия и мощността на обработка могат да възникнат от широкомащабни числени изчисления. Някои анализатори на СКА смятат, че тъй като компютърният хардуер напредва толкова бързо, няма нужда да се оптимизира производителността на СКА. Изследователите и софтуерните инженери обаче обикновено имат по-малка толерантност към дизайна на софтуер, който не е подходящ. Общата идея на техническите пазари е, че печалбите в производителността трябва да нарастват постоянно.

Извършването на изчисления на ръка отнема време, трудно е и е склонно към грешки. Започвайки преди приблизително 60 години, редица програми се опитаха да демонстрират, че е възможно да се отиде отвъд строго числовата област в научната област и да се използват за извършване на символно изчисление. Оттогава многобройни СКА са създадени от и

з В някои случаи СКА не успява да осигури точен резултат за някои м математически операции. Следователно трябва да имаме предвид, че тези с системи имат ограничения и не винаги са точни.

е Различните опростители отнемат различно време, за да завършат д един и същ математически израз. Например, опростителят *на Mathematica* е о приблизително 15 пъти по-бавен от *Symbolica (lib)* и опростители *на Maple* [5]. Някои малки СКА опростители (като Nemo) работят по-бързо от добре а познатите системи на Algebra. И така, скъпите търговски символни пакети т могат да бъдат заменени с надеждни, функционални безплатни символни и инструменти [6].

л ***Стандартите*** са от съществено значение за развитието на СКА. С Свободният поток от идеи и възможността да се гради върху успехите на другите винаги са били от полза за научните изследвания. Числовите



Т

е

з

и

библиотеки, стандартите *BLAS*, *IEEE с плаваща запетая* и *MPI* са примери за стандарти, базирани на най-добрите практики и колективния опит на по-голяма общност.

В компютърните науки, ***паралелизмът*** привлече много внимание, особено в дизайна на CPU архитектура, изкуствения интелект и езици за програмиране. Прилагането на множество парадигми за паралелно програмиране от СКА показва, че дори сложни алгебрични алгоритми могат да бъдат паралелизирани относително лесно. Например, различни паралелни методи могат да се използват от СКА за прилагане на алгоритми за решаване на системи с многовариантни нелинейни уравнения. Трябва да имаме предвид, че винаги има място за повече оптимизация. Например, Siegel (1993) постигна петкратно ускорение в сравнение с оригиналния последователен източник *на Maple* чрез паралелизиране на директно решение на сложния и важен проблем с истинската коренна изолация [8]. По-дългите времена за изчисление са резултат от повече математически операции. Според теорията на сложността, за най-добро време за изпълнение по-малкото операции са по-добри. Подобно правило се използва при високопроизводителни изчисления, когато всеки цикъл и инструкция са ценни. В едноядрената ера - която вече е еволюирала в многоядрената ера - това беше случаят. В исторически план математическите библиотеки са едни от първите приложения, които се адаптират към промените в хардуера с течение на времето. През последните 25 години е налице преминаване към повече многопроцесорни, разпределени и паралелни дизайнерски архитектури. Boehm и др. (2020) изследват различен метод за добавяне на паралелизъм към изчисленията в СКА. С техния ненатрапчив метод е възможно разпределеното изчисление. Тяхната основа е идеята, че координацията и изчислението трябва да се държат отделно. Високопроизводителната числена симулация вече успешно следва тази идея интерактивна програмна среда за паралелни изчисления. *Wolfram* интегрирана и автоматизирана среда. *Distributed Maple* е среда, предназначена да изпълнява програми за компютърна алгебра паралелно с хетерогенни клъстери и многопроцесорни системи. Schreiner (2003) описва архитектурата и употребата на околната среда. Системата използва език за програмиране *Java*, за да интегрира изчислителни машини от *Maple* в мрежов координационен слой [12].

***Преносимостта*** на кода на софтуерната библиотека винаги е била решаващ фактор, който трябва да се вземе предвид. Възможността да се скрият множество специфични за машината детайли в преносими софтуерни библиотеки позволява автоматизирана адаптация на платформата

за потребителя. Един пример за математически софтуерен пакет, който скрива машинните зависимости в модули от по-ниско ниво, като същевременно поддържа преносимост на най-високо ниво, е *LAPACK*. За приложните програмисти ***BLAS (набор от рутинни процедури на ниско ниво за извършване на общи линейни алгебрични операции) предлага преносим, ефективен и адаптивен стандарт, на който*** *LAPACK* силно разчита .

Различни техники за програмиране за СКА бяха сравнени от Frame и Coffey (2014). Констатациите показват, че докато функционалният подход предлага силно изразителни функции, които биха могли да улеснят разработката на софтуер, функционалните езици за програмиране вероятно не са подходящи за повечето математически интензивни приложения поради проблеми с производителността [14].

Студентите често използват *Python* и *Julia* , два невероятно гъвкави езика за програмиране, които ви позволяват да извършвате символни и числени изчисления. И двата са силни и имат предимства и недостатъци. Едно от основните предимства *на Julia* е неговата бързина. При паралелните изчисления сложните изчисления и големите набори от данни могат да отнемат време, така че скоростта е от решаващо значение. Кодът *на Python* е по-лек по време на изпълнение, но произвежда повече изход с по-бавно темпо. Освен това е по-добър вариант за анализ на данни и машинно обучение поради избора на библиотека. Въпреки това, *Julia* може да бъде по-добър избор за програмистите, когато се справят с изискващи изчисления задачи като статистически изчисления. Тоинт и Порчели (2021) заявяват, че съществуват софтуерни реализации за оптимизационни алгоритми със смесено цяло число без деривати и ограничени граници. Ограничените ситуации също могат да бъдат управлявани. Най-известните са написаните на C++ решаващи програми, като DAKOTA с отворен код, DFL решаващи програми и BFO.

Използването на изкуствени когнитивни системи (ACS) в обучението по математика също беше предложено, като теорията зад това е, че ACS могат да служат като инструменти за когнитивно преструктуриране. Тези ACS инструменти, като *Mathcad* , могат да помогнат за разбирането на произведените знания.

***Изкуственият интелект*** (ИИ) позволява на СКА да разбират ефективно математиката. Според скорошни изследвания производителността на СКА може да бъде подобрена чрез използване на техники за машинно обучение, като поддържащи векторни машини върху примерни проблеми. Освен това, иновативните употреби на СКА могат да бъдат резултат от нови прозрения в символното изчисление (предизвикани от обясними техники на ИИ) [20]. Една област, в която техниките на *машинното обучение МО* (ML) могат да се използват за оптимизиране или

избор на алгоритми, е символното изчисление. Huang (2016) заявява, че машинното обучение може да помогне при избора на СКА алгоритъм [21]. В опит да приложат ИИ директно към математиката, Lample и Charton (2020) обучиха трансформатор да интегрира изрази и да решава аналитично диференциални уравнения. Трансформаторът реши точно повече проблеми за определеното време от обикновените СКА [22]. През 2022 се дава обобщение на няколко инициативи, които използват машинно обучение за прогнозиране на свойствата на математическите структури [23]. Подобни констатации се показват в множество други проучвания, които подкрепят приложението на МО.

Вградените системи, които се намират най-вече в преносими устройства, съдържат по-голямата част от микропроцесорите, произвеждани днес. Тъй като последните се захранват от батерии, те трябва да работят на възможно най-ниската мощност. Високата консумация на енергия води и до други значителни проблеми, като разходите за охлаждане на системата поради произведената топлина. Правени са многобройни опити за оптимизиране на хардуера с цел намаляване на консумацията на енергия. Има проучвания обаче, които показват, че софтуерът представлява по-голямата част от консумацията на енергия на компютърната система [25].

**2.2 Общ изглед на СКА**

Shacham и Cutlip (1998) твърдят, че сравняването на математически софтуерни пакети гарантира избора на система, която напълно удовлетворява нуждите на всеки потребител. Но за да се направи сравнението, тези софтуерни програми трябва да бъдат изследвани с помощта на сравнителна перспектива, базирана на конкретни обективни стандарти. Изискванията са следните:

- Техническа поддръжка;
- Удобство за потребителя;
- Числено изпълнение.

Потребителските бази на *Mathematica, MATLAB* и *Maple* са големи, а техните интерфейси са лесни за използване. Това се подкрепя допълнително от факта, че уебсайтът на всяка компания и софтуерът имат интегрирани центрове за помощ. Всеки потребителски проблем може да бъде решен от дискусионните табла, ръководствата, документация и наличните видеоклипове. *Wolfram Mathematica* се занимава повече със символни изчисления, докато *MATLAB* се занимава повече с числени изчисления. Въпреки че са направени за различни видове задачи, и двата са силни инструменти. За потребители, които търсят добър инструмент предимно за инженерство и математика, *Maple* е най-добрият вариант. В сравнение с друг математически софтуер, като *MATLAB* и *Mathematica* , *Maple* не се поддържа толкова широко. *MATLAB* и *Mathematica* имат по-богати и по-активни общности от *Maple* .

В следващата таблица 1 можем да видим някои от основните характеристики на *Maple* , *Mathematica* и *MATLAB* . От тази таблица можем да заключим защо тези три СКА се класират най-високо и колко предизвикателство е да се избере само един.

**Таблица 1 .** Основни характеристики на Maple, Mathematica и MATLAB

| ***Параметри*** | | | |
|---|---|---|---|
| Удобен за потребителя | да | да | да |
| Осигурява ефективни и точни решения на сложни проблеми | да | да | да |
| Генериране на код на други езици за програмиране | Java, Perl, C#, Fortran, C, Python, Visual Basic и | C и C++ | B |
| Поддръжка на 2D обработка на изображения. | да | да | да |
| Поддръжка на обработка на 3D изображения. | Необходим е достъп до OpenGL библиотека за чертане на 3-D графики. | да | да |
| Възможност за редактиране на документи по време на изчисление | не | да | да |
| Спецификации на | Нуждае се от добра спецификация на | Нуждае се от добра спецификация на | Нуждае се от висока RAM |
| Използва се за | Изчисления в инженерството, квантовата химия, физиката и напредналата математика | Изчисления в математиката, инженерството, химията, физиката, биологията и други области | Изчисления в математиката, инженерството, химията, физиката, биологията, финансите и много други области |
| Цена | Високи (по-ниски | Някои пакети са | Високи (по-ниски |

| | | | |
|---|---|---|---|
| | цени за студенти) | достъпни безплатно за използване. | цени за студенти) |
| Най-доброто за | Потребители, които искат добър инструмент за математика и инженерство | Потребители, които искат да анализират данни и да ги моделират. Най-добре е да откриете измамни дейности. | Потребители, които искат помощ в невронни мрежи, моделиране на данни и визуализиране на симулации. |
| захранван код писане | не | не | Можете да съставите код, като опишете задачата с думи. |
| Автоматично довършване | да | да | да |
| Преформатиране на кода | не | не | да |
| Помощ за синтаксиса за | Липсва аргументи, обхват конфликти. | Липсващи аргументи, съвпадение в скоби, излишни аргументи. | Липсващи аргументи, съвпадение в скоби, излишни аргументи, конфликти в обхвата. |

**Глава 3**

В тази глава ще разгледаме някои важни и нововъзникващи технологии в системите за компютърна алгебра. В раздел **3.1** е представен изкуственият интелект.

3.1.1 Изкуствен интелект

Симулацията на интелигентно човешко поведение е известна като **изкуствен интелект** или накратко ИИ**.** Това е компютър или система, направена да усеща заобикалящата го среда, да разбира действията си и да реагира съответно на ситуации. Победата на суперкомпютъра на IBM “ *Deep* интерес към изкуствения интелект. Основите на ИИ са положени още в древността. Логиката се основава на метода за кодифициране на рационалната мисъл, който Аристотел е сред първите, които описват. Въпреки това, скокът за изкуствения интелект да бъде официална наука изисква известно ниво на математическа формализация. Така математиците

предоставиха инструментите и поставиха основите за разсъждения относно алгоритмите и разбиране на изчисленията. Въвеждането на изкуствения интелект в науката беше само въпрос на време. Това важи и за математиката, която в момента е една от най-популярните, фундаментални и „трудни“ науки. За добро или лошо, човешкият ум е ограничен. В този случай технологията се използва, за да тласне човечеството напред, независимо дали в областта на автоматизацията и практическите въпроси или знанието и изследването. Една от основните цели на изкуствения интелект е да създава модели. За да се подготвят тези модели, могат да се прилагат идеи и техники от различни математически дисциплини. Помислете за автомобилите Arduino, чиято цел е да идентифицират препятствията. Тези автомобили се захранват от математика, по-специално *линейна алгебра*, *смятане* и *теория на игрите*.

Изкуственият интелект, който се основава на алгоритми, съществува в живота ни в много по-голяма степен, отколкото си представяме. Можем да видим това, когато играем шах срещу компютъра в *Chess.com*, настолни игри с виртуален опонент и футболни игри като *FIFA 2024* срещу виртуален играч.

През 2021 г. математиците успяха да си сътрудничат с изкуствен интелект за разработване и валидиране на нови математически теореми за първи път. Университетът в Оксфорд, университетът в Сидни в Австралия и по проекта. В сравнение с генерираните от хора версии, разработените алгоритми могат да сортират данни до три пъти по-бързо. Резултатите от това проучване предполагат, че машинното обучение може да подпомогне изследванията в областта на математиката! [27]

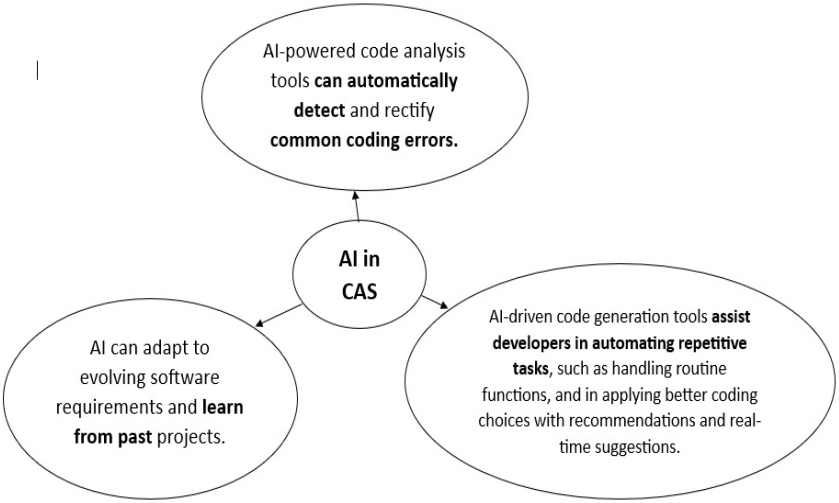
В допълнение към опитите за разбиране, изкуственият интелект също има за цел да автоматизира и организира умствените процеси. ИИ, който дава на машините способността да изпълняват изчислителни и други задачи, подобни на човешките способности, е създаден чрез комбиниране на машинно обучение с големи данни. Използването на изкуствени невронни мрежи и опит за подобряване на когнитивното поведение е известно като ***машинно обучение*** . В резултат на това данните захранват ИИ, а ИИ ботовете могат ефективно да изпълняват всяка задача, която може да бъде изследвана в данни. Дори в ситуации, в които корелацията не е очевидна веднага, тези системи свързват сложни процеси и анализират огромно количество данни. Големите размери на извадката могат да се използват за идентифициране на модели в машинното обучение, което след това изгражда модел, който включва тези модели и позволява на модела да се „*учи*“. Моделът става по-предсказуем, когато се извършват хиляди повторения на този процес.

Светът се променя драстично в епохата на четвъртата индустриална революция благодарение на технологиите, чиито основни играчи са интернет, облачните изчисления, големите данни, роботиката и изкуственият интелект. Добре известните математически алгоритми са отговорни за всички тези постижения. Ето защо е много вероятно много скоро да живеем в нов свят, съчетан с машини и роботи. Освен това шансовете ни за успех в тези нови условия се увеличават с по-математически ориентиран подход. Фигура 1 илюстрира как ИИ-разширението влияе върху посоката на развитие на СКА, а таблица 2 показва защо СКА трябва да премине към ИИ.

И така, типичните предимства на ИИ са: а*)* **Намаляване процента на човешка грешка.** *б)* **По-бързо вземане на решения и действия.** *в)* **Идентифициране на тенденции и правене на бъдещи прогнози.**

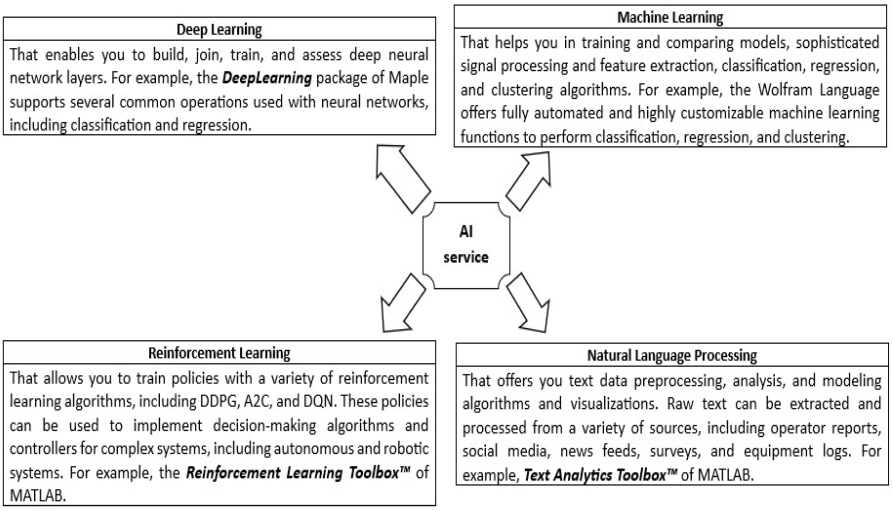
**Таблица 2 Защо СКА трябва да премине към ИИ**

| **Традиционни СКА** | **СКА с ИИ технология** |
|---|---|
| Те са прост компютърен софтуер, който манипулира символни изрази и може да се използва по начин, подобен на ръчните изчисления. | Те са компютърен софтуер, който симулира човешки мисловни процеси за извършване на сложни задачи като обучение, разсъждение и анализ. |
| Използвайте манипулиране на символни изрази, за да извършите точна математика. | Използвайте статистически методи, които извличат правила от данни. |
| Алгоритмите често включват решения и изчисления, които могат значително да повлияят на необходимите ресурси ( *използване на памет, използване на процесор* и т.н.), без да добавят нищо към решаването, въпреки че са изложени на повече данни. | Той автоматизира процеса на създаване на аналитични модели, които извличат данни за скрити прозрения, използвайки ИИ техники (напр. *невронни мрежи* и др.). Тъй като тези алгоритми са изложени на повече данни, те се представят по-добре с времето. ИИ може да използва предварително дефинирани правила; не винаги се нуждае от големи набори от данни. |
| Обикновено има един или няколко метода за решаване, които не се променят с времето. | Те включват различни методи (като *обучение с подсилване* , *неконтролирано* и *контролирано обучение* и т.н.), които подобряват решаването с течение на времето. Чатботовете могат да решават основни математически проблеми или да ви дават съвети за решаването им. |




**Фигура 1 .** Начините, по които подобрената с ИИ разработка влияе върху посоката на развитие на СКА

На следващата фигура 2 можем да видим аналитично какъв вид ИИ услуги предоставят.




**Фигура 2**

### 3.1.2 Математика и ChatGPT

Най-известната система за диалог с въпроси и отговори е Chat технология е голям пробив в областта на генерирането на код. Инструментът спестява време и повишава производителността, като помага на разработчиците да автоматизират различни аспекти на разработката на софтуер. *ChatGPT* беше заменен от *GPT-4* през март 2023 г. Докато вторият може да се използва и като инструмент за обикновени математически проблеми, първият е полезен за търсене на математически термини, техники, алгоритми и т.н. [29].

## 3.2 Паралелно изчисление

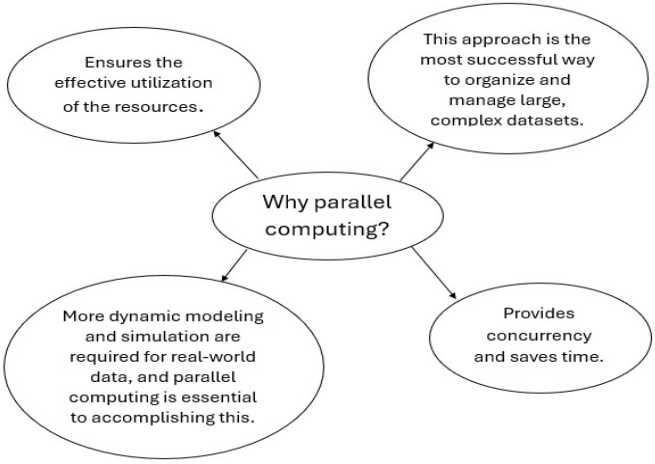
Използването на няколко процесора наведнъж за решаване на изчислителен проблем е известно като ***паралелно изчисление***. За програмистите и математиците е трудна задача да намерят алгоритми за управление на комуникацията и синхронизацията между множество подзадачи, които могат да се използват паралелно. Въпреки че математическата общност е направила значителни крачки в решаването на тези проблеми, паралелизирането, например в символното изчисление, остава трудно за математиците и разработчиците на софтуер. Не всички задачи могат да бъдат ефективно паралелизирани и ефективността на паралелното изчисление често зависи от вида на проблема, който се решава. В повечето случаи паралелизирането съкращава общото време до завършване. За съжаление, не всички процесори получават еднакъв дял от изчислителното натоварване и много от тях са неактивни за продължителни периоди. Да се надяваме, че тъй като размерите на процесорите продължават да растат, ще трябва да се изпълняват по-малко инструкции от системата за обработка на големи количества данни. Например, две инструкции са необходими за добавяне на две 16-битови цели числа в 8-битов процесор. За разлика от това, в 16-битов процесор е необходима само една инструкция.

Най-лесният и най-евтиният начин за извършване на паралелизиране е *SIMD* (единична инструкция с множество данни), която е налична на почти всички процесори. Чрез процес, известен като *авто-векторизация*, компилаторите могат автоматично да създават *SIMD* инструкции за различни цикли, без да изискват въвеждане от потребителя. Има изключителен *SIMD* регистър, който може да зарежда няколко стойности една след друга наведнъж. И така, *SIMD* инструкциите обработват множество части от данни за разлика от обикновените инструкции.

Паралелизмът трябва да бъде включен в СКА, за да се възползвате напълно от съвременните многоядрени процесори и високопроизводителни клъстери. Целта е да се реши проблем възможно най-бързо, като се използват възможно най-много компютърни ресурси, вместо серийно изчисление. На фигура 3 можем да видим защо математическата наука

трябва да премине към паралелни изчисления. Създаването на необходимите условия за писане на паралелен код в СКА е важна задача, засягаща всеки системен аспект. Налични са два общи модела за паралелни изчисления. Докато първият модел изпълнява множество независими процеси, които трябва да уведомяват един или повече от другите процеси за техния напредък, вторият модел позволява на множество нишки да извличат една и съща информация от паметта. Така че за приложения, включващи разпределено или паралелно изчисление, препоръчителната практика за програмиране е първо да се определи дали изисква независима или комуникационна работа. Второ, трябва да се избере най-ефективният начин за разделяне на независима работа на задачи.

Например, разделянето на частичните диференциални уравнения на поддомейни, които могат да се обработват независимо от различни процесори, **би било ефективна стратегия**. Алгоритъмът трябва да бъде проектиран така, че да сведе до минимум комуникацията между процесорите и да подобри производителността, като гарантира, че зависимостите на данните са регионални. Изборът на подходящ програмен модел, като популярния **OpenMP** (за системи със споделена памет) или **MPI** (за разпределени системи), е важен, за да помогне на процесорите да комуникират и да се координират един с друг. Като гарантират, че изчислителните задачи се изпълняват правилно в няколко ядра или възли, тези рамки спомагат за максимизиране на производителността. И накрая, ефективното използване на изчислителни ресурси ще доведе до оптимални времена за изпълнение, ако внедряването е профилирано и настроено въз основа на реална хардуерна производителност.




**Фигура 3.** Защо паралелни изчисления?

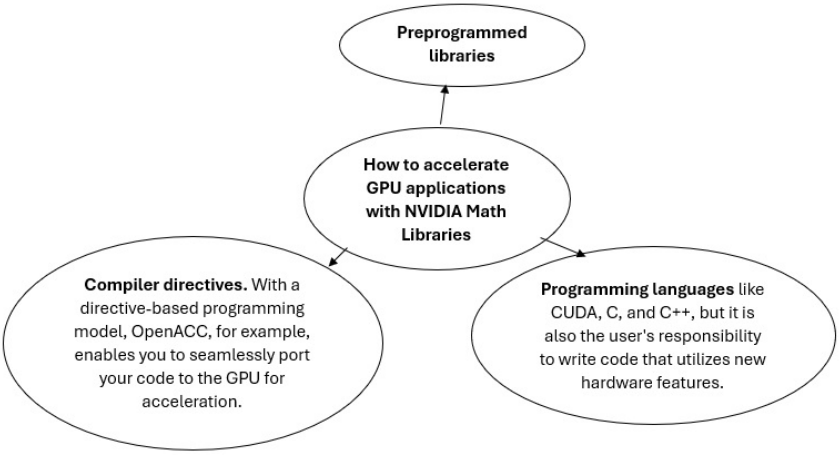
Тъй като много известни приложения и проблеми от реалния свят са моделирани чрез **линейни системи**, решаването им е един от най-значимите проблеми в научните изчисления. Тези системи често са много големи, което прави решаването им особено времеемко. Това е причината ефективното паралелизиране на тяхното решение да стане от съществено значение за получаване на значителни резултати за разумен период от време. Така че систематичното разработване и прилагане на паралелни алгоритми ще бъде основна задача през следващите години.

**Итеративните/приблизителните** и **систематичните методи** са двата известни начина които са използвани за решаване. Последните методи са по-бързи за изпълнение, по-подходящи за плътни матрици на коефициенти и винаги дават решения на линейната система чрез ясни математически изчисления. Итеративните/приблизителните методи са подходящи за редки масиви и обикновено се изпълняват много по-бързо (което е една от основните причини да бъдат разработени и използвани на практика), но не винаги са полезни поради тяхната несигурна конвергенция.

Основата на паралелното изчисление на езика Wolfram е способността да се стартират и управляват множество процеси на ядрото на езика Wolfram от един главен език Wolfram. Езикът Wolfram предоставя много начини за паралелно изпълнение на работници, или локално на една и съща машина, или отдалечено през мрежа. Освен това мрежата може да има хомогенна мрежа или разнородна мрежа под контрола на конкретно приложение за управление.

Тъй като модулите за **графична обработка могат да управляват хиляди нишки** наведнъж, те стават все по-популярни, особено за симулации. NVIDIA и AMD, двата най-големи производители на графични карти, предлагат SDK за създаване на приложения за своите графични процесори. Известният NVIDIA HPC ( **High** - **P**erformance **C**omputing) SDK библиотеки на NVIDIA, които осигуряват отлични реализации на функции, намиращи се в различни приложения с интензивни изчисления. Само с незначителни модификации на кода, тези библиотеки са предназначени да ускорят приложенията на NVIDIA GPU и да заменят популярни CPU библиотеки като OpenBLAS, LAPACK и Intel MKL. На следващата Фигура 4 виждаме как можем да ускорим GPU приложения с NVIDIA Math Libraries.

За да извършат **общо умножение** на матрица, **NVIDIA** GPU разделят изходната матрица на плочки, които впоследствие се разпределят към блокове с нишки. За да изчисли своята изходна плочка, всеки блок от нишки зарежда необходимите стойности от двете матрици ( *Mtile x Ntile* ), умножава ги и ги събира в изхода и след това преминава през измерението K в плочки




**Ф**

Технологията, известна като **H**igh-**P**erformance **Computing** (HPC), обработва големи, многоизмерни н**а**бори от данни и решава сложни проблеми с невероятно бързи скорост**у**и, като използва клъстери от мощни и ефективни процесори, които работят **р**паралелно. На същото високо ниво е **и квантовото изчисление** , което реш**а**ава сложни проблеми чрез използване на специализирана технология, базирана на квантовата механика.

Изчислителната ефективност на много ИИ операции е напълно трансформирана от възможностите на GPU. Процесите на обучение и извеждане на моделите за дълбоко обучение са значително ускорени от технологии като CUDA и Tensor Cores на NVIDIA, които позволяват оптимизирано изпълнение на матрични операции. Паралелните изчисления ще революционизират начина, по който работят СКА в бъдеще. С много устройства, постоянно свързани помежду си, нарастващите разпределени системи и бъдещите по-бързи мрежи, сигурно е, че паралелните изчисления ще играят решаваща роля след няколко години.

**3.3 Големи обеми от данни**

Технологичното развитие с оперативна съвместимост и цифровизация като доминиращи елементи води до производството на огромни обеми данни с висока скорост. Тези огромни обеми от данни, **Big** разнообразие, които изискват ефективни и иновативни форми на обработка на информация. Тези три фактора – обем, скорост и разнообразие – станаха известни като 3Vs (обем, скорост, разнообразие) на Big Data.

Потопът от данни засяга почти всеки сектор на икономическата и социална дейност, докато през следващите години се прогнозира още по-голяма „информационна експлозия“. Обществото на информацията и

знанието, в което живеем, се основава на тази ценна съставка, данните. Данните обаче са суровината. Техният анализ (Big Data Analytics) е необходим, за да се получи добавена стойност. По-специално, използването на големи данни, в смисъл на извличане на надеждни и полезни резултати, изисква подходяща процесорна мощност, наличие на организирана изчислителна инфраструктура, зони за съхранение (например използване на възможностите за съхранение, предлагани от облака) и инструменти за анализ. Поради това беше създадена необходимостта от нови математически методи и практики, тъй като съществуващите решения за управление на данни не отговориха напълно на този голям обем.

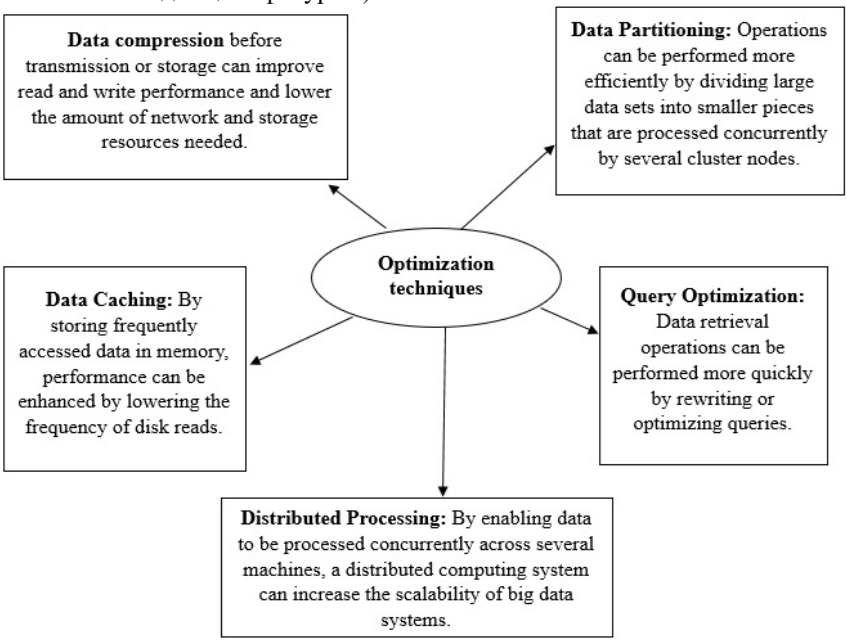
Днес, благодарение на мощните процесори и алгоритми, тези данни могат да бъдат анализирани. Възможността за съхраняване, обобщаване, комбиниране на данни и използване на резултатите за изготвяне на подробни анализи вече е достъпна и постижима. Средствата за извличане на знания от данни (Data Mining) непрекъснато се подобряват, тъй като наличният софтуер за прилагане на тези техники се комбинира с нарастващата изчислителна мощност.

Работата с Big Data се улеснява от СКА, които се свързват и интегрират със съхранението на Big Data и се адаптират към вашите изисквания за обработка на данни въз основа на наличните ресурси. Например *MATLAB* може да има достъп до големи количества от данни от различни системи за съхранение, да прилага и разширява един и същ код, без да променя алгоритмите, да анализира и създава модели за машинно обучение върху по-малки количества данни и в крайна сметка да използва мощността за обработка, следвайки изискванията на потребителя. **Паралелното обучение** може да бъде полезно при обучение на мрежа с големи количества данни. Използването на няколко мини-партиди наведнъж по време на обучение може да съкрати времето, необходимо за обучение на мрежа. В *MATLAB* се препоръчва обучението да се извършва с един или повече GPU. *Maple* използва алгоритми за дълбоко обучение с TensorFlow на Big Data. *Mathematica* предлага набор от инструменти за най-широк анализ и визуализация с интегриран изчислителен интелект.

Извличането на полезна информация от Big Data е една от основните цели на Statistics. Статистиката предлага рамка за събиране, оценка и интерпретиране на големи данни, което я прави решаващ компонент на науката за данни и машинното обучение. Например, процесът на **почистване на данни** е една област, в която статистиката играе критична роля в науката за големите данни чрез премахване на неподходящи данни и дубликати, филтриране на липсващи стойности и т.н.

**Оптимизацията на големи данни** или процесът на подобряване на големи данни за по-голяма точност, ефективност, полезност и уместност е една от основните концепции на анализа на големи данни. Примери за тези

методи включват намаляване на времето за изпълнение на задачите и необходимите ресурси (като памет и съхранение). ***Сортирането и почистването*** на големи данни е стъпка в процеса на оптимизация на данните, която подобрява качеството и уместността на данните чрез елиминиране на грешки, несъответствия и излишъци. Системите за големи данни могат да работят по-добре, когато използват различни стратегии за оптимизация, като *разделяне на данни* , *компресиране* и *кеширане* (подробно описание в следващата фигура 5).




**Фигура 5.** Някои техники за оптимизация за Big Data

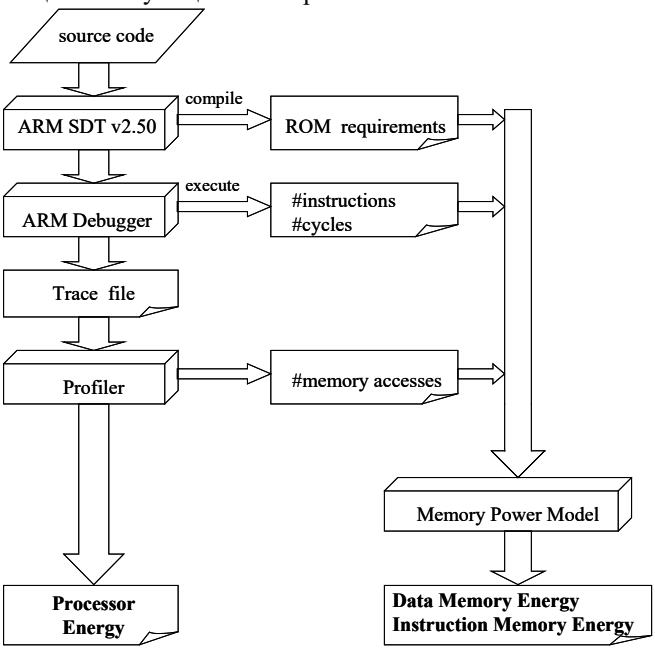
### 3.4 Енергийна сложност и потребление

Вградените системи, които се намират най-вече в преносими устройства, съдържат по-голямата част от микропроцесорите, произвеждани днес. Тъй като последните се захранват от батерии, те трябва да работят на възможно най-ниската мощност. Високата консумация на енергия води и до други значителни проблеми, като разходите за охлаждане на системата поради произведената топлина. Правени са многобройни опити за оптимизиране на хардуера с цел намаляване на консумацията на енергия. Има проучвания обаче, които показват, че софтуерът представлява по-голямата част от консумацията на енергия на компютърната система.

Тъй като повече хора използват преносими устройства, които трябва да използват по-малко енергия, широко се приема, че ниската консумация

на енергия е важна. Тъй като контролира дейността в основната схема, работещият софтуер оказва значително влияние върху енергията в изчислителната система. Същото важи и за СКА, които сега са в мобилни телефони като приложения.

Концепцията за **енергийна сложност** на алгоритъм за изчисляване на необходимата консумация на енергия е представена от автора на тази дисертация в много статии с много математически примери (алгоритми за умножение на матрици, редове на Фибоначи, алгоритъм на Хилберт, проблем на Collatz и други). ***За да се оцени въздействието на достъпа до паметта на данни и инструкции върху общата консумация на енергия, се въвежда концепцията за енергийната сложност на алгоритъма (и консумацията на енергия)***. Следната фигура 6 демонстрира как може да се изчисли общата консумация на енергия.

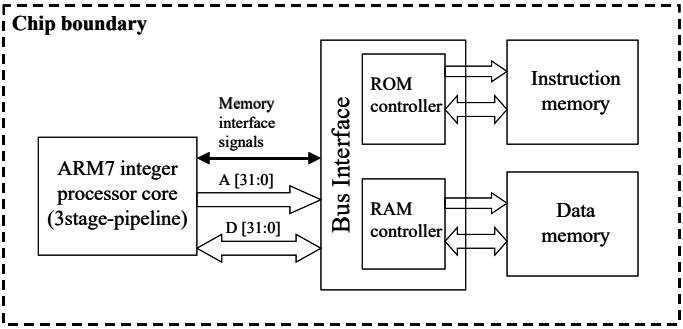



**Фигура 6 .** Създаден експеримент за оценка на консумацията на енергия

Беше взета под внимание обобщена целева архитектура, за да се оцени предложената енергийна сложност (Фигура 7). Приехме общ модел на изчисление с един процесор и машина с произволен достъп (RAM) като технология за внедряване. Поради своята обещаваща MIPS/mW производителност, ядрото на процесора с цяло число ARM е популярен избор за вградени приложения. Компилирането на всеки *C++* код с

помощта на компилатора на ARM Developer Suite е първата стъпка в процедурата, която беше следвана при провеждането на експериментите.

Разсейването на енергия от алгоритъм може да се опише чрез неговата енергийна сложност, която е сравнима с неговата изчислителна сложност. В допълнение към изразите за броя на примитивните операции или инструкции, които са били изпълнени, целта е да се извлече полиномиален израз за броя на *достъпите до паметта за данни* . Консумацията на енергия от *достъпите до паметта на данни* може лесно да се изчисли, тъй като всеки достъп до паметта има известна (количествено измерима) цена на енергия.




**Фигура 1.** Целева архитектура

От друга страна, енергията, използвана в *процесора* и *паметта на инструкциите,* може да бъде извлечена с помощта на полиномиални изрази на броя на примитивните операции. Тези два енергийни компонента се получават лесно, тъй като броят на достъпите до паметта на инструкциите е равен на изпълнените инструкции за асемблиране и всяка примитивна операция приблизително се преобразува в независима инструкция за асемблиране, чиято консумация на енергия може да бъде изчислена. Следователно общата енергийна сложност ще бъде определена, както следва:

$$E_{total} = c_1 \cdot E_{proc} + c_2 \cdot E_{instr\_mem} + c_3 \cdot E_{data\_mem}$$

where:

$c_1 \cdot E_{proc}$ corresponds to the processor energy and is a polynomial expression of the number of primitive operations times a coefficient $c_1$. Coefficient $c_1$ corresponds to the average energy consumed during the execution of an assembly instruction and can be accurately estimated from physical power measurements and profiling of representative applications.

$c_2 \cdot E_{instr\ mem}$ corresponds to the instruction memory energy and is a polynomial expression of the number of primitive operations times a coefficient $c_2$. Coefficient $c_2$ corresponds to the energy cost of an access to the instruction memory.

$c_3 \cdot E_{data\ mem}$ corresponds to the data memory energy and is a polynomial expression of the number of memory accesses times a coefficient $c_3$. Coefficient $c_3$ corresponds to the energy cost of an access to the data memory.

The coefficients $c_1$, $c_2$, and $c_3$ are dependent on the computer system used and $E_{total}$ is the total calculated energy complexity of the algorithm under study.

***Забелязва се, че използването на енергийна сложност за изчисляване на потреблението на енергия е по-близо до реалното потребление, отколкото използването на общата изчислителна сложност за изчисляването му.*** Освен това се наблюдава, че извикването *на функция* причинява висока консумация на енергия (целевият експеримент беше серията *на Фибоначи*). Същото се случва с решенията *за програмиране на шаблони на дизайн* (енергийно скъпи) в сравнение с решенията за дизайн *без шаблони* .

**Глава 4**

В тази глава се разглеждат някои съвети и техники, които подобряват производителността на СКА**.** Инструментите за сравнителен анализ често се използват за измерване на производителността на дадена система. Еталон в изчислителната техника е процесът на изпълнение на компютърна програма за оценка на производителността (време, консумация на енергия, използване на памет и използване на процесор). Той също така покрива производителността на хардуера и мрежата, ако е от съществено значение. Изчисленията обикновено се изпълняват на един и същ хардуер за всички програми, които се сравняват.

Сравнителният анализ на компютърната алгебра се различава от другите задачи за сравнителен анализ по много начини. Първо, изчислителните резултати не винаги са уникални. Множество изходни данни, които не са точно равни на пръв поглед, все още могат да бъдат еквивалентно правилни. Второ, често е трудно да се определи дали даден отговор е правилен, ако не е уникален. Трето, компютърната алгебра обхваща широк спектър от теми, следователно са необходими набори от показатели. Четвърто, във всяка СКА начинът за решаване на математически проблем се различава по отношение на алгоритъма за изпълнение и когато става въпрос за входни формати, много системи за компютърна алгебра следват различни пътища. В областта на компютърната алгебра добре познат бенчмарк е ***DEval,*** който е инструментариум за сравнителен анализ, създаден върху базата данни със символни данни. ***StarExec*** е друг известен инструмент, но той е специално проектиран за общности, които използват логически решаващи програми и основната му цел е да управлява библиотеки за сравнение чрез платформа.

Густаво и Тот (2019), използвайки вградени скриптове за сравнителен анализ, установиха, че *Mathematica* има предимство в символните изчисления в сравнение с *MATLAB*. От друга страна, *MATLAB* има по-добра ефективност с големи числови данни, по-ориентирана е към данни и е идеална за проектиране на функции. Решаване на линейна система, числено интегриране и транспониране на матрици са някои от темите, за които *MATLAB* има по-добро време в сравнение с *Mathematica*. Но въпреки

тези резултати, сравняването им е по-трудно, отколкото изглежда на пръв поглед, поради фундаменталните разлики в начина, по който са структурирани**,** как се изчисляват крайните резултати и колко значими цифри се използват. *Mathematica* е по-мощна със системата за програмиране на високо ниво, особено за символни манипулации и много полезна за работа със смятане и диференциални уравнения. Освен това, тя предлага по-задълбочена, по-всеобхватна и по-добре интегрирана колекция от функции в своите основни алгоритми. Функционалността *на Mathematica* често е само частично внедрена в *Maple*. Wolfram *Maple* е известен със способността си да решава много ефективно сложни уравнения.

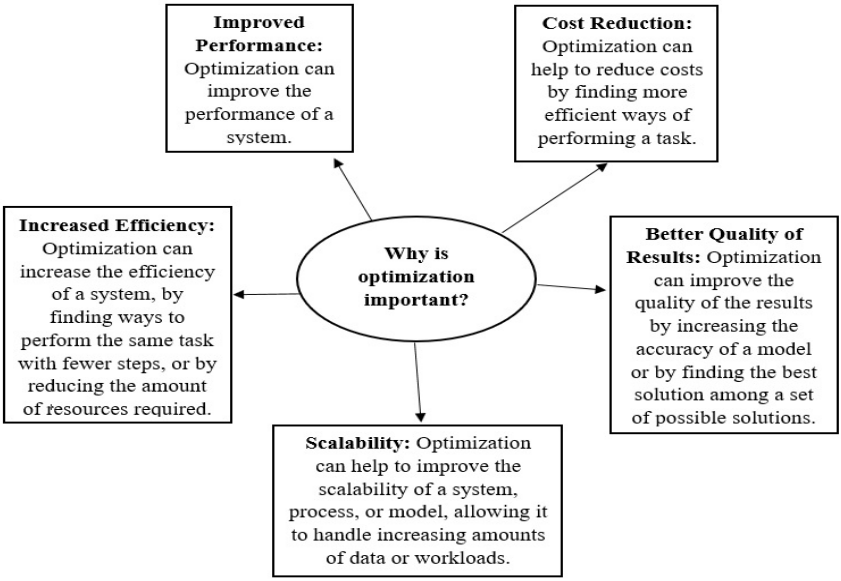
**4.1 Съвети за по-добро представяне на СКА**

Накъдето и да погледнете, от интернет маршрутизирането до инженерния дизайн, от разписанието на авиокомпанията до финансите, оптимизацията присъства. ***Изчислителната оптимизация*** се отнася до направата на системата - хардуер или софтуер - да работи възможно най-бързо и стабилно. Тъй като времето, парите и ресурсите са важни, оптимизирането на системата е много по-важно.

Оптимизацията е важна, защото може да повиши ефективността и да подобри качеството на резултатите, като същевременно спомага за подобряване на производителността на системата чрез избор на оптималната опция от набор от жизнеспособни решения. Оптимизирането на производителността на софтуера е от полза по много причини, както можем да видим на фигура 8.

Колекцията от най-добри практики и техники, използвани за запазване на върховата производителност на СКА, е известна като **СКА оптимизация.** Този раздел разглежда няколко стратегии и тактики, които подобряват ефективността на СКА**.** Повечето от съветите са за това как да програмирате в тези системи и какво трябва да знаете, преди да ги използвате. Известните СКА са *многопарадигмени*, което означава, че можете да ги използвате за *процедурно*, *императивно*, *функционално* или *обектно-ориентирано* програмиране. Въпреки че някои се интерпретират и оптимизират от *JIT* технология за компилиране, която елиминира например излишния код и пренарежда командите, все пак тези инструкции, представени тук, са важни и приложими с многостранни ползи за потребителя на СКА.

***Предварителното разпределение е от полза.*** Предварителното разпределение намалява разходите за разпределение на паметта, като гарантира, че матричните елементи ще бъдат запазени в съседни местоположения в паметта само веднъж. Така че, ако знаете предварително размера на масива, можете да създадете "*празен*" масив с този размер като начало. Това ще направи вашия скрипт по-бърз за големи набори от данни и няма да има почти никакъв ефект върху малки набори от данни.




**Фигура 8.** Защо оптимизацията е важна?

Използването и производителността на паметта могат да бъдат повлияни от цикли, тъй като те увеличават размера на структурата от данни всеки път през цикъла, карайки СКА многократно да се опитва да побере структурата от данни в по-големи съседни блокове памет. Паметта може да стане фрагментирана поради динамично разпределение и освобождаване на памет. Таблица 3 показва стратегии за ефективно използване на паметта.

**Таблица 3** Стратегии за ефективно използване на паметта

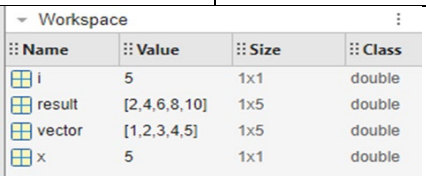
| *(Избягвайте)* | *Следвайте)* |
|---|---|
| Създайте големи временни променливи | Изчистете временните променливи, когато вече не са необходими. |
| Създайте свои функции за изчистване на паметта | Задайте променливи, равни на празния масив, за да освободите памет или изчистете променливите с помощта на вградени функции. |
| Създайте нови променливи | Използвайте повторно променливите колкото е възможно повече |
| Преоразмерете масива всеки път, когато го увеличите. | Предварително разпределяне |

Излишно е ***извършването на операция вътре в цикъл, когато тя е независима от индекса на цикъла.*** Вградените цикли могат да бъдат неефективни и да отнемат много време, тъй като повтарят един и същ набор от данни многократно. Проблеми с производителността могат да възникнат с нарастването на размера на данните, тъй като може да отнеме

експоненциално повече време за повторение на данните. Сложността на алгоритмите може да бъде намалена от $O(N^2)$ до $O(N)$ или дори по-ниска, като се избягват вложени цикли.

***Препоръчват се векторни операции вместо цикли*** (вижте таблица 4 за пример). Векторизиращият код може да подобри производителността, тъй като повечето СКА извършват матрични и векторни изчисления, използвайки оптимизирани за процесора библиотеки.

**Т**

| | |
|---|---|
| **а***% Ето един прост пример за цикъл*<br>**б**x = 5;<br>**л**вектор = 1:x;<br>**и**резултат = нули (1, x);<br>**ц***% цикъл*<br>**а**за i = 1:x<br>  резултат(i) = вектор(i)*2;<br>**4**край | *% Същото и с векторизацията*<br><br>вектор = 1:x;<br>*% векторизация*<br>резултат = вектор.*2; |
| Прост пример за ве |  |



***Добра практика е да се използват строги правила за опростяване***. Някои СКА имат алгебрични функции за опростяване. Например, в *изразът* е масив или символен вектор, тази функция опростява всеки елемент от *израза* . Вижте примера в таблица 5. Внимавайте, защото в някои случаи ***резултатите от математическите изрази може да се различават от оригиналния израз след опростяване***. Крайният израз може да не е пропорционален на началния израз. Например, правилото за опростяване *на IgnoreAnalyticConstraints* в *MATLAB* с *истинска* стойност може да причини този проблем.

***В повечето случаи профилирането е полезно, но е по-загрижено за ефективността, отколкото за точността.*** Например, в *Maple*, използването на *profile()* увеличава използването на паметта и има някои проблеми с процедури, които имат специални правила за оценка. MATLAB разпознаем кои функции прекарват най-много време, ще можем да ги променим. Също така, можем да профилираме нашия код, като изолираме части от кода, за да решим кои редове от код консумират повече компютърни източници и да локализираме "*горещите точки*". В таблица 6 има пример

за профилиране в *Octave*, а фигура 9 демонстрира цикъла от стъпки за оптимално профилиране.

**Таблица 5**

```
>> S = опростяване (exp(b*log(sqrt(ac))))
```

Алгебрично опростяване

Workspace

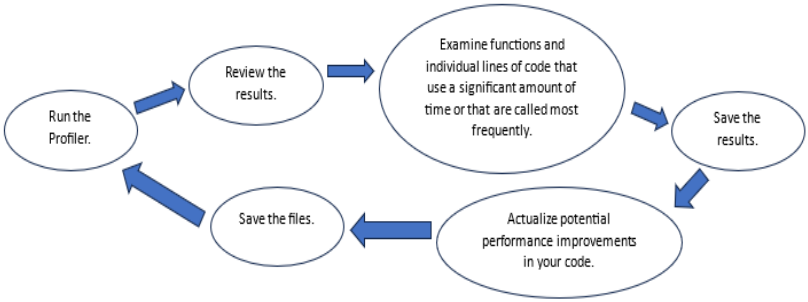
| Name | Value | Size | Class |
|---|---|---|---|
| S | (a - c)^(b/2) | 1x1 | sym |
| a | a | 1x1 | sym |
| b | b | 1x1 | sym |
| c | c | 1x1 | sym |
| x | x | 1x1 | sym |

**Таблица 6**

```
профилът е включен
Символен pkg v3.0.1: Активна комуникационна връзка на Python, SymPy v1.9.
профилът е изключен
данни = профил ("информация");
profshow (данни, 10);
```

Пример

```
         #                Function Attr     Time (s)    Time (%)         Calls
-------------------------------------------------------------------------------
        79                  system              1.336       54.49             3
       121                   pause              0.986       40.22            67
        74       python_ipc_popen2              0.013        0.53             2
        16 double_to_sym_heuristic              0.013        0.52             1
       116                readblock              0.012        0.49             5
        17                  assert              0.010        0.39           154
         1               fibonacci              0.007        0.27             1
       125    extractblock>helper    R        0.006        0.26             7
       160   display>sym_describe              0.006        0.25             1
       135                 num2str              0.006        0.24             1
```




**Фигура 26**

***Препоръчва се вградените функции да не се претоварват.*** Може да има предупреждение в определени СКА, което ви предупреждава за наличието на вътрешна функция. Повечето от тях ви позволяват да използвате оригиналната вградена функция, въпреки че сте я претоварили. Можем да видим пример в *Octave* в Таблица 7.

**Таблица 7. Претоварване на функцията Tan(x) и извикване след това с *вградена* команда**

```
функция y = tan (x), y = cos (x); крайна функция
октава:2> тен(0)
ивграден ("тен", 0)
```

***Функциите, използващи една и съща променлива като вход и изход, са с предимство***. Ще можем да намалим броя на дубликатите в нашия създаден код, като напишем функции, които използват една и съща променлива като вход/изход. В таблица 8 представя пример в *MATLAB*.

**Таблица 8. Практика на кодиране за оптимизиране на референтни параметри**

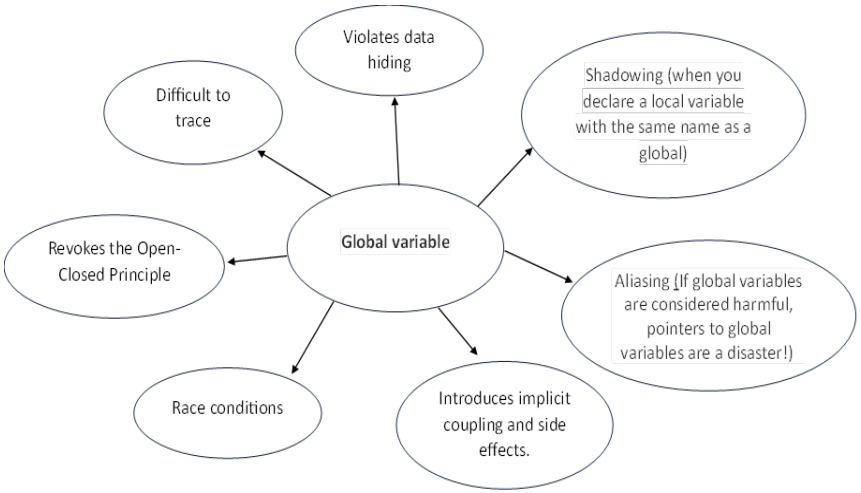
```
функция Y = example1(X,Y)
Y = X - Y;
край
```

В горния пример *Y* действа като вход и изход, генерираният код предава стойността на *Y* ***чрез препратка,*** вместо многократно да репликира *Y* към временна променлива.

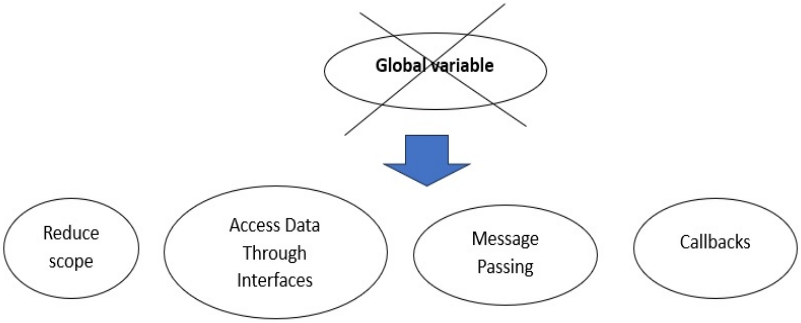
***Глобалните променливи са опасни.*** Глобалната променлива се декларира извън всяка функция и е достъпна за всички рутинни процедури. Това прави кодирането по-лесно, но също така създава проблеми с ***модулността*** и ***гъвкавостта*** на програмата. Освен това всяка промяна в техните стойности се разпространява в цялата програма. Фигура 10 изобразява проблемите с глобалните променливи, а фигура 11 показва някои техники за елиминиране на глобалните променливи.

***Функциите са за предпочитане пред скриптовете.*** Има СКА, които поддържат както функции, така и скриптове. Въпреки че не са еквивалентни, те ви позволяват да използвате повторно последователността от команди, които сте въвели, като ги записвате във файлове. Функциите са по- ***адаптивни*** и ***разширяеми*** от скриптовете, така че са по-добър избор.

За един програмист писането на ясен, разбираем и поддържаем код е основна способност. Чистият код означава, че е лесен за разбиране и промяна (Фигура 12 обяснява какви принципи правят кода лесен за разбиране, а Таблица 9 показва за какво трябва да се погрижим).




**Фигура 2.** Проблемите с глобалните променливи




**Фигура 3.** Техники за премахване на глобалните променливи

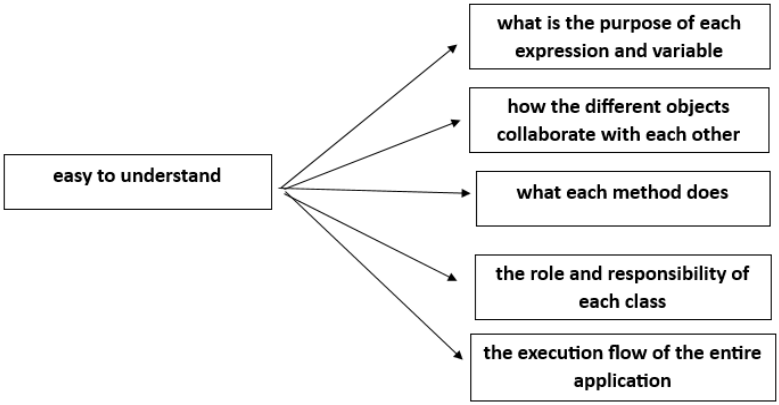
Причината, поради която трябва да избягваме генерирането на ***боклук*** (разпределено хранилище, което не е необходимо), е, че това увеличава натоварването на събирача на боклук и удължава времето, необходимо за завършване на задачата. Ето защо трябва да изхвърлим (или да зададем на боклук. В *MATLAB* и много други СКА мениджърът на паметта оптимизира, за да избегне фрагментирането на куп след деструктори на обекти. ***Мързеливата инициализация*** е един полезен метод за създаване на обекти само когато е необходимо. Има много други техники за подобряване на времето за изпълнение и намаленото използване на паметта на генерирания код, например, някои СКА използват постоянно сгъване, сливане на цикъл, елиминиране на недостижим код и други подобни техники за оптимизация.

Какво трябва да използвате и избягвате, когато пишете код

| Използвайте | Избягвайте |
|---|---|
| Произносими и смислени имена | Шумни думи |
| Една дума за всяко понятие | Флаг аргументи |
| По-малко аргументи | Странични ефекти |
| Дефиниране на нещо веднъж и само веднъж | |
| Ръководства за стил, специфични за езика | |
| Коментари | |

***Поддържането на нещата организирани*** да се предотврати загуба на данни, също е добра практика. За да запазите свързаните файлове заедно, дайте на всеки проект своя папка. Използвайте коментари, особено коментари в заглавието. Избягвайте използването на рискови команди в скрипт, като например командата *MATLAB clear all* .

***Неизползваният код*** трябва да бъде премахнат от нашите скриптове по много причини. Първо, всеки нов потребител трябва да разбере не само работния код, но и неизползвания материал. Това е объркващо и е загуба на време. Второ, възможно е в даден момент да бъде направена промяна от друг потребител, който неволно да повлияе на „*спящия*“ код, което може да доведе до грешки.

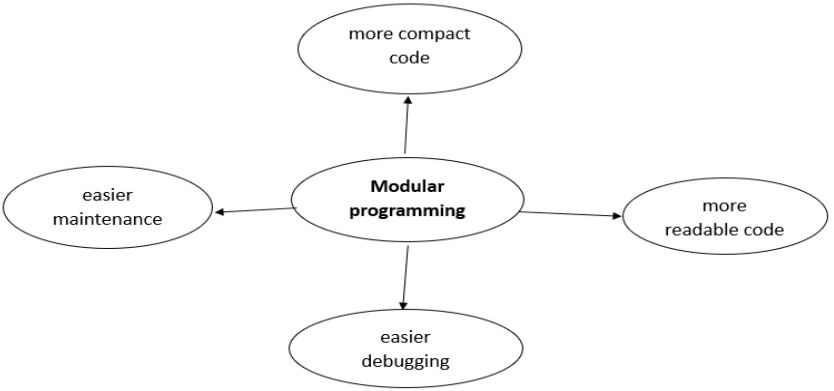



**Фигура 4.** Какви принципи правят кода лесен за разбиране

***Модулното програмиране има много предимства.*** За да го постигнете, разделя програмните функции на независими части ***подпрограми***, ***методи*** *или* ***процедури*** . Скриптът е неструктурирана кодова последователност в един файл **.** На фигура 13 можем да видим предимствата на модулния дизайн на програмата.

***операторите*** са по-полезни (например *И* и *ИЛИ*). Философията зад поведението на тези къси съединения е, че не е необходимо да изчислявате

втория израз, ако вече знаете резултата. Например, логическата операция *И* използва такова поведение. Когато първият израз е *FALSE,* няма смисъл да се изчислява вторият израз. Можем също да проверим това от случаите *3* и таблица 10.




**Фигура 5.** Ползите от използването на модулно програмиране

**Таблица 10**

| Логично И | | | |
|---|---|---|---|
| **Опция** | **Израз 1** | **Израз 2** | **Резултат** |
| | ВЯРНО | НЕВЯРНО | НЕВЯРНО |
| | ВЯРНО | ВЯРНО | ВЯРНО |
| | НЕВЯРНО | ВЯРНО | НЕВЯРНО |
| | НЕВЯРНО | НЕВЯРНО | НЕВЯРНО |

В следващата таблица 11 можем да видим пример за MATLAB. В този пример е безсмислено да се оценява връзката отдясно. Първата релация се оценява на *логическо 0* (FALSE), така че изразът създава късо съединение.

**Фрагмент 10**

```
Var1 = 5;
Var2 = -2;
Резултат = (Var1+1 > 12) && (Var2+Var1 < -6)
```

```
Резултат = логическо 0
```

***Изборът на правилния алгоритъм и тип данни е от решаващо значение.*** Например, когато работим върху версия на алгоритъма на Дейкстра, алгоритъмът се ускорява чрез превключване от *списък* към *купчина* .

***Когато е възможно, използвайте редки масиви***, тъй като те могат да спестят памет и време за изчисление. Само ненулевите/нулеви елементи трябва да се съхраняват в разреден масив, тъй като по-голямата част от елементите имат една и съща стойност и стойността по подразбиране на

масива е *null* или *нула* . Използването на рядко представяне може да бъде много полезно, ако конкретна стойност се среща често. В *Mathematica* със вектори и тензори.

***Изчисленията могат да се ускорят чрез извършване на паралелни изчисления***. Процесът на разделяне на задание на части и изпълнението им едновременно на няколко процесора или ядра е известен като ***паралелно изчисление.*** По този начин производителността може да бъде подобрена и общото време за изчисление може да бъде намалено. Въпреки че възникват няколко проблема, когато математическите алгоритми се решават паралелно, паралелизмът трябва да бъде включен в СКА колкото е възможно повече, за да се възползваме напълно от съвременните многоядрени процесори и високопроизводителни клъстери. Целта е да се реши проблем възможно най-бързо, като се използват възможно най-много компютърни ресурси, вместо серийно изчисление. В *MATLAB* паралелната обработка се извършва основно чрез кутия с инструменти. Това ви позволява да използвате паралелно активирани функции, да управлявате *NVIDIA® GPU* директно от *MATLAB* с *gpuArray*, да изпълнявате паралелни итерации на *for-цикъл (parfor)* и да изпълнявате множество симулации едновременно, като използвате цялата процесорна мощ на многоядрените настолни компютри. Както се споменава в документацията *на MATLAB*, трябва да внимаваме с паралелната обработка, тъй като циклите се изпълняват без ред и това означава, че нашият код трябва първо да бъде модифициран, за да съответства на това изпълнение. Освен това една итерация на цикъл не трябва да зависи от предишна итерация *(много важно!)*.

Паралелна обработка се извършва в *MAPLE* по два начина, с *модела за програмиране на задачи* (инструмент за програмиране на високо ниво) и с *пакета Grid.* В първия имаме много задачи в един процес, а във втория много процеси са активирани в началото. Езикът Wolfram (*Mathematica*) предлага силна и отличителна среда за паралелни изчисления, която ви позволява да паралелизирате автоматично, да споделяте памет, да използвате многопроцесорни процесори и да извършвате многоядрени изчисления.

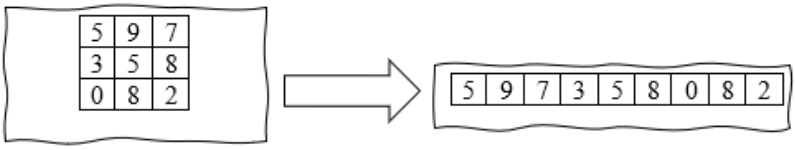
***Сортирането и почистването на Big Data*** е стъпка в процеса на оптимизация на данните, която подобрява качеството и уместността на данните чрез елиминиране на грешки, несъответствия и излишъци. Системите за големи данни могат да работят по-добре, когато използват различни стратегии за оптимизация, някои от които са: *а) компресиране* и *кеширане на данни*, *б) оптимизиране на заявки* и *в) разделяне на данни*.

Тъй като повече хора използват преносими устройства, които трябва да използват по-малко енергия, широко се приема, че ***ниската консумация на енергия е важна***. Тъй като контролира дейността в основната схема,

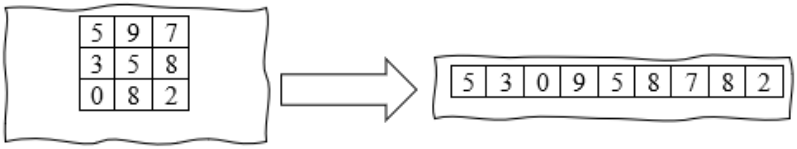
***работещият софтуер оказва значително влияние върху енергията в изчислителната система.*** Същото важи и за СКА, които сега са в джобовете ни като приложения. В някои математически парадигми се забелязва, че използването на енергийна сложност за изчисляване на потреблението на енергия е по-близко до реалното, отколкото използването на общата изчислителна сложност. Освен това се наблюдава, че извикването *на функция* причинява висока консумация на енергия. Същото се случва с решенията *за програмиране на шаблони за дизайн* (енергийно скъпи) в сравнение с решенията за дизайн *без шаблони* .

***В масивите по-бързо е да се сканират колони надолу, отколкото редове*** (това зависи от езика за програмиране, който използваме). От *реда на главната колона* се подразбира, че докато елементите по ред са разделени в RAM, елементите по колона са последователни. Ефективността на кеша се увеличава чрез сканиране на колони, въпреки че времевата сложност и на двете е една и съща (Например, за *2D* таблици е $O(n^2)$). Фигура 14 показва двата типични метода (*главен ред на ред* и *основен ред на колона*) за навигация в масив.

- **row-major-order**




- **column-major-order**

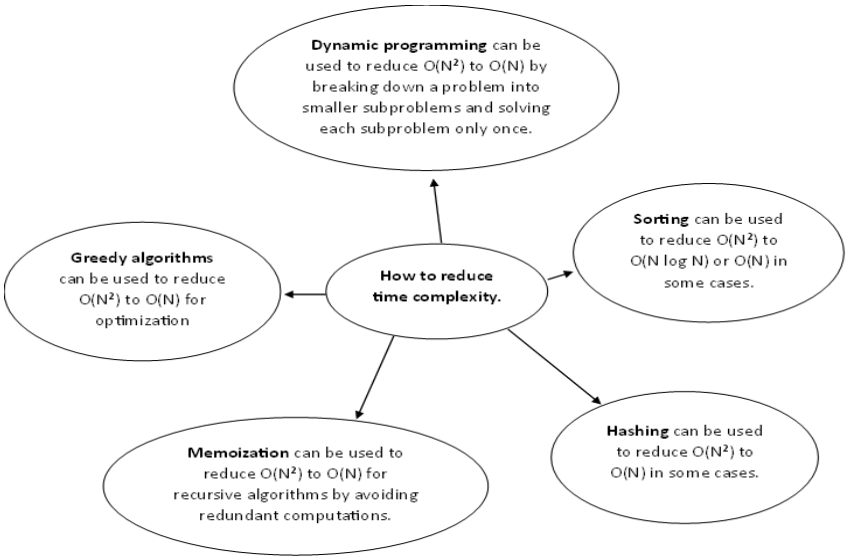



**Фигура 6.** Основен ред на ред *срещу* основен ред на колона

Можете да използвате онлайн версията на СКА, ако вашият компютър не отговаря на минималните спецификации за настолната версия (RAM, процесорна мощност и т.н.). Можете да правите много математически изчисления направо във вашия браузър и да работите с други потребители на СКА, като им предоставите необходимите разрешения и директно споделяте файлове. Можете да синхронизирате критичните си математически файлове с Cloud Drive и да съхранявате файлове там. Освен това можете да използвате най-новата версия на СКА, без да се налага да я изтегляте, инсталирате или поддържате.

***Хеширане, сортиране, динамично програмиране, алчни алгоритми*** **и** мемоизация (техника за оптимизация), която елиминира излишните изчисления – всички те могат да намалят времевата сложност на математическите алгоритми. Фигура 15 илюстрира тези методи за намаляване на изчислителната сложност на математическите алгоритми.

Обърнете внимание на съветите, предоставени от ***Code Analyzer.*** Докладът на анализатора на код показва възможни грешки и проблеми заедно с възможности, базирани на съобщения, за подобряване на вашия код. По подразбиране проблемите са групирани въз основа на сериозността. С **филтрите** можете да стесните съобщенията, които се показват. Докато някои проблеми могат да бъдат автоматично заменени, други проблеми могат да бъдат разрешени чрез избор на подходящия избор.




**Фигура 7.** Общи техники за намаляване на времевата сложност на алгоритмите

За справяне с всеки математически проблем, много системи за компютърна алгебра предоставят разнообразие от евристики, процеси на вземане на решения и настройки на параметри; въпреки това потребителите трябва ръчно да изберат тези опции, за да използват СКА. Тези решения могат да окажат значително влияние върху жизнеспособността или дори ефективността на решаването на проблема. Тъй като няма определени правила, които да диктуват оптималния курс на действие и тъй като дори за експертите често има малка връзка между разглеждания проблем и избора на алгоритъм, ние често сме принудени да разчитаме на машинно обучение.

***Машинно обучение*** е името на областта на компютърните науки, която изучава създаването на алгоритми, които „*учат*“, без да бъдат програмирани със специфични правила. С други думи, тези алгоритми използват данни, за да открият модели и връзки, за да направят прогнози или да вземат решения. СКА трябва да приеме подходи на ИИ, защото*:*

*a)* Рационализира процеса на разработване на аналитични модели, които използват техники (като невронни мрежи) за извличане на данни за неоткрити знания;

*b)* Включва набор от техники, които подобряват решаването на проблеми с течение на времето, включително *обучение с подсилване, контролирано* и *неконтролирано обучение* и др.

***Измерванията показват забележително подобрение на производителността, когато се използват тези техники.*** *Предварителното разпределение* и *векторизацията*, например, могат да ускорят кода с няколко порядъка, твърди *Mathworks*. Новите алгоритми в защото предварително разпределят памет и се изпълняват на място.

**Глава 5**

Използвайки софтуер за символно изчисление, математици, учени, инженери и преподаватели могат да се справят със сложни компютърни изчисления. Има много причини, като например възможността за концептуално свързване на математически проблеми с проблеми от реалния свят, визуализиране на математически проблеми, лесен достъп от всяко място и гъвкавост.

Онлайн системите за компютърна алгебра станаха популярни сред преподаватели и студенти през последните десет години. Има многобройни предимства, включително повишена гъвкавост, лекота на използване, достъпност от всяко място и т.н. Те обаче имат някои недостатъци, които трябва да знаем.

Ние непрекъснато се опитваме да оптимизираме всяко инженерно и промишлено приложение, независимо дали става дума за максимизиране на печалбата, производителността, производителността или ефективността, или за минимизиране на разходите и потреблението на енергия. *Изчислителната оптимизация* се отнася до това да накарате система - хардуер или софтуер - да работи възможно най-бързо и стабилно. Тъй като времето, парите и ресурсите са важни, оптимизирането на системата е много по-важно.

*Оптимизацията на системата за компютърна алгебра (СКА)* се отнася до колекция от методи и най-добри практики, използвани за поддържане на оптималната работа на СКА. Най-добрите практики се отнасят до това как настройвате система, пишете скриптове или

изпълнявате някаква математическа задача. Оптимизирането на производителността на СКА в еднаква степен във всяка среда е изключително предизвикателство. Производителността е силно повлияна от различни фактори, включително *памет*, *операционна система* и *процесорна мощност*. Изискванията, характеристиките и ограниченията на различни устройства, платформи и уеб браузъри може да варират и да окажат влияние върху функционалността на СКА. Докато някои от техниките, обсъждани тук, са специфични за конкретни СКА, по-голямата част от тях са общи и се прилагат за всички СКА.

**5.1 Резултати**

В този раздел е представено обобщение на всички техники, за да ви помогне да оптимизирате производителността при използване на СКА. По-голямата част от това съдържание е общо и приложимо за всички СКА. Ето защо, следвайте тези съвети, ако искате да се представите по-добре:

• ***В повечето случаи профилирането е полезно, но е по-загрижено за ефективността, отколкото за точността.*** Като разпознаем кои функции работят най-дълго време, ще можем да ги променим. Също така, можем да профилираме нашия код, като изолираме части от кода, за да решим кои редове от код консумират повече компютърни ресурси и да локализираме "*горещи точки*". Това ръководство може да е най-важното от всички.

• Обърнете внимание на съветите, предоставени от ***Code Analyzer.*** Докладът на анализатора на код показва възможни грешки и проблеми заедно с възможности, базирани на съобщения, за подобряване на вашия код.

• За да изберете правилния математически инструмент, ***уверете се, че вашият проблем и неговите цели са ясни*** .

• ***Добра практика е да се използват строги правила за опростяване***. Внимавайте, защото в някои случаи резултатите от математически изрази може да се различава от оригиналния израз след опростяване. Крайният израз може да не е пропорционален на началния израз.

• ***Препоръчват се векторни операции вместо цикли.*** Векторизираният код може да подобри производителността, тъй като повечето СКА извършват матрични и векторни изчисления, използвайки оптимизирани за процесор библиотеки.

• ***Модулното програмиране има много предимства.*** По-компактен и четим код, по-лесна поддръжка и отстраняване на грешки **.**

• Излишно е ***извършването на операция вътре в цикъл, когато тя е независима от индекса на цикъла.*** Вградените цикли могат да бъдат неефективни и да отнемат много време, тъй като повтарят един и същ набор от данни многократно. Проблеми с производителността могат да възникнат

с нарастването на размера на данните, тъй като може да отнеме експоненциално повече време за повторение на данните. Сложността на алгоритмите може да бъде намалена от *O(N²)* до *O(N)* или дори по-ниска, като се избягват вложени цикли.

• ***Препоръчва се вградените функции да не се претоварват.***

• ***Функциите, използващи една и съща променлива като вход и изход, са с предимство***. Ще можем да намалим броя на дубликатите в нашия създаден код, като напишем функции, които използват една и съща променлива като вход/изход. ***Глобалните променливи са опасни.*** Глобалната променлива се декларира извън всяка функция и е достъпна за всички рутинни процедури. Това прави кодирането по-лесно, но също така създава проблеми с ***модулността*** и ***гъвкавостта*** на програмата. Ако типът на данните се промени, по-добре е да създадете нови променливи.

• ***Когато е възможно, използвайте редки масиви***, тъй като те могат да спестят памет и време за изчисление. Само ненулевите/нулеви елементи трябва да се съхраняват в разреден масив, тъй като по-голямата част от елементите имат една и съща стойност и стойността по подразбиране на масива е *null* или *нула*. Използването на рядко представяне може да бъде много полезно, ако конкретна стойност се среща често.

• ***Предварителното разпределение е от полза.*** Предварителното разпределение намалява разходите за разпределение на паметта, като гарантира, че матричните елементи ще бъдат запазени в съседни местоположения в паметта само веднъж. Така че, ако знаете предварително размера на масива, можете да създадете "*празен*" масив с този размер като начало. Това ще направи вашия скрипт по-бърз за големи набори от данни и няма да има почти никакъв ефект върху малки набори от данни. Използването и производителността на паметта могат да бъдат повлияни от цикли, тъй като те увеличават размера на структурата от данни всеки път през цикъла, карайки СКА многократно да се опитва да побере структурата от данни в по-големи съседни блокове памет.

• Тъй като повече хора използват преносими устройства, които трябва да използват по-малко енергия, широко се приема, че ***ниската консумация на енергия е важна***. Тъй като контролира дейността в основната схема, ***работещият софтуер оказва значително влияние върху енергията в изчислителната система.***

• ***Функциите са за предпочитане пред скриптовете.*** Има СКА, които поддържат както функции, така и скриптове. Въпреки че не са еквивалентни, те ви позволяват да използвате повторно последователността от команди, които сте въвели, като ги записвате във файлове. Функциите са по-***адаптивни*** и ***разширяеми*** от скриптовете, така че са по-добър избор.

• Можете да използвате онлайн версията на СКА, ако вашият компютър не отговаря на минималните спецификации за настолната версия

(RAM, процесорна мощност и т.н.). Можете да правите много математически изчисления направо във вашия браузър и да работите с други потребители на СКА, като им предоставите необходимите разрешения и директно споделяте файлове.

• ***Хеширане, сортиране, динамично програмиране, алчни алгоритми* и** мемоизация **—** техника за оптимизация, която елиминира излишните изчисления — всички те могат да намалят времевата сложност на математическите алгоритми.

• ***В масивите е по-бързо да се сканират колони надолу, отколкото редове*** (това зависи от езика за програмиране, който използваме). От *реда на главната колона* се подразбира , че докато елементите по ред са разделени в RAM, елементите по колона са последователни. Ефективността на кеша се увеличава чрез сканиране на колони, въпреки че времевата сложност и на двете е еднаква.

• ***Изчисленията могат да се ускорят чрез извършване на паралелни изчисления***. Паралелното изчисление използва множество компютърни ядра за извършване на изчисление едновременно. По този начин производителността може да бъде подобрена и общото време за изчисление може да бъде намалено. Въпреки че възникват няколко проблема, когато математическите алгоритми се решават паралелно, паралелизмът трябва да бъде включен в СКА колкото е възможно повече, за да се възползвате напълно от съвременните многоядрени процесори и високопроизводителни клъстери. Целта е да се реши проблем възможно най-бързо, като се използват възможно най-много компютърни ресурси.

• ***Сортирането и почистването на големи данни*** е стъпка в процеса на оптимизация на данните, която подобрява качеството и уместността на данните чрез елиминиране на грешки, несъответствия и излишъци. Системите за големи данни могат да работят по-добре, когато използват различни стратегии за оптимизация, някои от които са: *а) компресиране* и *кеширане на данни*, *б) оптимизиране на заявки* и *в) разделяне на данни*.

• ***Операторите за късо съединение*** са по-полезни (например *И* и *ИЛИ* ). Философията зад поведението на логическото късо съединение е, че не е необходимо да изчислявате втория израз, ако вече знаете резултата.

• За един програмист писането *на ясен*, *разбираем* и *поддържаем* код е основна способност. ***Чистият код*** означава, че е лесен за разбиране и промяна.

Причината, поради която трябва да избягваме генерирането на ***боклук*** (разпределено хранилище, което не е необходимо), е, че това увеличава натоварването на събирача на боклук и удължава времето, необходимо за завършване на задачата. Ето защо трябва да изхвърлим (или да зададем *null*)

всичко, от което вече не се нуждаем, така че да може да бъде взето като боклук.

***Мързеливата инициализация*** е един полезен метод за създаване на обекти само когато е необходимо. Има много други техники за подобряване на времето за изпълнение и намалено използване на паметта на генерирания код, например някои СКА използват постоянно сгъване, сливане на цикъл, елиминиране на недостижим код и други подобни техники за оптимизация.

***Поддържане на нещата организирани***, за да се предотврати загуба на данни, също е добра практика. За да запазите свързаните файлове заедно, дайте на всеки проект своя папка. Използвайте коментари, особено коментари в заглавието. Избягвайте използването на рискови команди в скрипт. ***Неизползваният код*** трябва да бъде премахнат от нашите скриптове по много причини. Първо, всеки нов потребител трябва да разбере не само работния код, но и неизползвания материал. Това е объркващо и е загуба на време. Второ, възможно е в даден момент да бъде направена промяна от друг потребител, който неволно да повлияе на „*спящия*“ код, което може да доведе до грешки.

• ***Изборът на правилния алгоритъм и тип данни е от решаващо значение.*** Например, когато работим върху версия на алгоритъма на Дейкстра, алгоритъмът се ускорява чрез превключване от *списък* към *купчина*.

• Докато традиционният софтуер все още може да генерира основни прогнози въз основа на исторически данни, той не е толкова предсказуем, колкото *машинното обучение*. Дори ако алгоритмите са изложени на повече данни, те често включват изчисления и отговори, които могат да имат значително въздействие върху необходимите ресурси (памет, процесор и т.н.), без да добавят нищо към проблема. Потребителите на СКАтрябва да възприемат подходите на ИИ, защото*: а)* Той рационализира процеса на разработване на аналитични модели, които използват техники за извличане на знания от данни; *б)* Включва набор от техники, които подобряват решаването на проблеми във времето.

Интегрирането на ИИ в СКА фундаментално променя начина, по който подхождаме към решаването на проблеми. Тази иновация бележи промяна към по-персонализирано и интуитивно потребителско изживяване и има силата да разсее напълно нашите предубеждения за това какво може да постигне СКА. ***Взаимодействието на потребителя с СКА се променя значително, подготвяйки сцената за бъдещето, в което тези системи ще могат да идентифицират и да се адаптират към отделните предпочитания и поведение на всеки потребител.***

### 5.2 Предимства и недостатъци от използването на онлайн СКА

В този параграф ще обобщим предимствата и недостатъците на онлайн СКА. Следователно с онлайн СКА можете да:

- Вземате най-новата версия на СКА с всички най-нови налични функции, без необходимост от изтегляния, инсталации или поддръжка;
- Използвате браузъра си, за да правите всякакви математически изчисления;
- Извършвате математически операции, дори ако вашата машина не отговаря на минималните изисквания на настолната версия;
- Синхронизирате вашите критични компютърни файлове с СКА Cloud Drive и съхранявайте файлове там;
- Използвате файловете офлайн;
- Имате достъп до вашите онлайн СКА файлове, дори ако сте инсталирали различна операционна система на вашия компютър или сте форматирали твърдия диск;
- Работите заедно с други потребители на СКА, като директно споделяте файлове и им предоставяте правата, които желаете. Вие контролирате кой има достъп до четене, писане, стартиране и работа с вашите файлове;
- Имате връзка, така че хората да могат да намерят вашия проект в социалните медии. Цялото ви СКА съдържание може незабавно да се сподели;
- Използвате интерактивни функции направо във вашия браузър;
- Добавяте видеоклипове и друго онлайн съдържание направо във вашите бележници;
- Отваряте файлове, които сте направили с настолната версия на СКА;
- Използвате безплатен софтуер с отворен код, като Octave, който позволява на всеки да го използва, променя и разпространява, стига да признае за всички модификации, направени съгласно условията на оригиналния лиценз;
- Подготвите и анализирате данни, без да пишете код, като използвате интуитивен интерфейс;
- Имате възможност вашите проекти да са достъпни от всеки компютър с интернет връзка.

Сред основните недостатъци на онлайн СКА са следните:

- В сравнение с онлайн версията, десктоп СКА предлага повече опции за поверителност и сигурност, както и вероятно по-добра производителност (в зависимост от спецификациите на системата);
- Постоянно се нуждае от интернет връзка и влизане;

- Онлайн версиите имат ограничения. Например, определен хардуер, като контрол на инструменти, не може да се използва с онлайн MATLAB;
- Някои допълнителни инструменти за опаковане не съществуват;
- Качването на много големи файлове изисква Cloud Drive;
- Графичният потребителски интерфейс се различава при настолните версии;
- Поддържат се по-малко продукти, отколкото при настолните версии;
- Понякога е необходим абонамент;
- Десктоп версията съхранява всички данни на вашия компютър, така че вие сте единственият с разрешение да ги *преглежда, чете, пише* и *споделя* за разлика от онлайн версията. Кибератаките не са заплаха и не се генерират бисквитки. Уеб приложенията създават повече уязвимости в сигурността.

Системите за онлайн компютърна алгебра изглежда са много известни сред преподаватели и студенти. Те могат да бъдат достъпни от всяко устройство, преносими са и са по-лесни за използване. Техните иновации премахват недостатъците на предишните версии, което води до среда за програмиране, която е лесна за използване и продуктивна. Новите възможности обхващат много други научни области в допълнение към математиката. Можем само да се надяваме, че с това развитие недостатъците на настоящите онлайн системи - като неподдържан хардуер или липса на инструменти - ще бъдат разрешени и новите платформи ще могат да правят повече в бъдеще.

От друга страна, настолните СКА предлагат по-добра производителност, мрежова независимост и повече опции за поверителност и сигурност. Те могат лесно да взаимодействат с хардуера и не са ограничени по никакъв начин. СКА за настолни компютри трябва да се инсталират ръчно и актуализират и им липсва гъвкавостта на онлайн СКА. Освен това изискванията, функциите и ограниченията на различните устройства и операционни системи може да варират, което може да окаже влияние върху функционалността.

## Глава 6

### 6.1 Приноси

В този дисертационен труд беше направено цялостно подробно представяне на наличните СКА (десктоп и онлайн) с всички характеристики, като година на първо издание, употреба, цена, дистрибуторска компания и др. Бяха идентифицирани съществуващите проблеми (например в онлайн СКА допълнителните инструменти за

опаковане не съществуват, поддържат се по-малко продукти, отколкото с настолните версии и т.н.) и предимствата бяха подчертани (например онлайн СКА са по-лесни за използване). Беше направено сравнение между настолни компютри и онлайн версии и бяха подчертани ползите, които всяка от тях има за потребителя. На тази основа беше направено наблюдение на най-известните СКА(въз основа на литературата и измервания, направени от независими изследователи) и беше установено, че някои СКА се представят по-добре и са по-подходящи за някои математически операции.

Беше подчертано значението на паралелната обработка. Производителността може да бъде подобрена с паралелизиране и общото време за изчисление може да бъде намалено значително. Въпреки че възникват няколко проблема, когато математическите алгоритми се решават паралелно, паралелизмът трябва да бъде включен в СКА колкото е възможно повече, за да се възползвате напълно от съвременните многоядрени процесори и високопроизводителни клъстери. Целта е да се реши проблем възможно най-бързо, като се използват възможно най-много компютърни ресурси, а не серийно изчисление

Беше подчертана стойността на потреблението на енергия. Енергийната сложност за изчисляване на потреблението на енергия е по-близка до реалната, отколкото при използване на общата изчислителна сложност. Освен това се наблюдава, че има много техники и съвети, които можем да следваме, за да увеличим максимално ефективността, производителността и ефективността. Тези инструкции понякога могат да подобрят до голяма степен СКА изпълнение.

Живеем в нова ера, в която изкуственият интелект навлезе в живота ни до голяма степен, това неизбежно се отрази и на СКА. Например, разбирането как да се реши даден математически проблем сега може да стане с решения стъпка по стъпка, а не със старомодното незабавно представяне на математически резултати. Интегрирането на ИИ в СКА фундаментално променя начина, по който подхождаме към решаването на проблеми и тази теза подчертава нейното значение.

6.1.1 Научен принос

Колко добре работи СКА зависи много от това как я използваме. Тази теза подчертава, че няма значение с коя СКА работим, а какви настройки правим, как програмираме и какви команди използваме. Бяха представени повече от четиридесет инструкции, повечето от които приложими за всички СКА. Вярвам, че би било много полезно тези съвети да бъдат включени в бъдещите СКА под формата на съвети (докато потребителят въвежда командите). Потребителите на СКА трябва да знаят тези съвети, ако искат да подобрят производителността и времето за изпълнение на своите програми. Освен това, това изследване значително помогна да се даде на

математическата общност повече основания по въпросите на настройките на СКА, математическото програмиране и управлението на кодове.

Има много повече неща, които СКА трябва да може да направи. Тук бяха разкрити необходимостта от по-лесни за използване и по-интелигентни СКА, достъпни отвсякъде (чрез интернет, мобилни телефони и т.н.) и способни да се адаптират към изискванията на потребителите чрез проучването на повечето СКА. Освен това беше предложено използването на ИИ за математически проблеми и беше подчертано значението на паралелната обработка.

6.1.2 Научни и приложни приноси

В изследването беше направено сравнение на СКА по отношение на технически характеристики (като изисквания за RAM и т.н.), възможности (като обработка на *3D* изображения и преформатиране на код), цена, дали са удобни за потребителя и дали осигуряват ефективни и точни решения на сложни проблеми. Въз основа на тези сравнения бяха направени заключения. Освен това, чрез представяне на измервания, направени в проучвания от независими изследователи, беше направено общо сравнение на известните СКА една с друга. Разбира се, беше подчертано, че в тези системи, поради специалната архитектура, която имат и различните приложения, които изпълняват, е много трудно да се избере само една.

Беше 2005 г., когато енергийната сложност беше спомената за първи път от автора на тази дисертация (с над 30 цитата на статии, публикувани в известни списания). Много други публикации с експерименти върху потреблението на енергия от различни математически алгоритми други) последваха този документ. Целта беше да се представи по-точен начин за изчисляване на консумираната енергия във вградените системи и да се подчертае важността на отчитането не само на времевата сложност на кода, но и на енергията, която се консумира. В наше време СКА съществуват на нашите телефони като приложения. Компаниите отчаяно се опитват да намерят начини да удължат живота на батерията, така че тази информация беше ценна не само за математическата общност, но и за програмистите на вградени системи, които се опитват да намерят начини да намалят консумацията на батерията.

### 6.2 Публикувани статии, свързани с темата на дисертацията

ostas Zotos and Irina Athanasova, 2023, Advantages & Disadvantages of Online СКА. Journal of Software Engineering and Applications, Vol.16,

7. Kostas Zotos, 2008, Computer Algebra Systems - New Strategies and Techniques. Elsevier - Journal of Applied Mathematics and Computation, 198 (2008) 123–127.

**Извадка на използваната литература**

Survey of User Interfaces for Computer Algebra Systems. Journal of Symbolic Computation (1998) 25, 127–159.

ahn, L. Ristau (2020). Towards Massively Parallel Computations in Algebraic Geometry. Foundations of Computational Mathematics (2021) 21:767–806. https://doi.org/10.1007/s10208-020-09464-x
le: parallel Computer Algebra in networked environments. Journal of Symbolic Computation 35 (2003) 305–347.

Davies, A., Veličković, P., Buesing, L. et al (2021). Advancing Mathematics by guiding human int