СПЕЦИАЛИЗИРАН НАУЧЕН СЪВЕТ ПО ИНФОРМАТИКА И МАТЕМАТИЧЕСКО МОДЕЛИРАНЕ ПРИ ВАК

БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО ПАРАЛЕЛНА ОБРАБОТКА НА ИНФОРМАЦИЯТА

Гергана Иванова Бенчева

ПАРАЛЕЛНИ АЛГОРИТМИ ЗА РАЗДЕЛЯНЕ НА ПРОМЕНЛИВИТЕ И ФАКТОРИЗАЦИЯ НА РАЗРЕДЕНИ МАТРИЦИ

Д И С Е Р Т А Ц И Я

за присъждане на

образователната и научна степен „Доктор“
Научна специалност: 01.01.09 „Изчислителна математика“

Научен ръководител:
ст.н.с. д-р Панайот Василевски
Научен консултант:
ст.н.с. I ст. дмн Светозар Маргенов

София, 2004

Съдържание

Списък на фигурите

Списък на таблиците

Увод

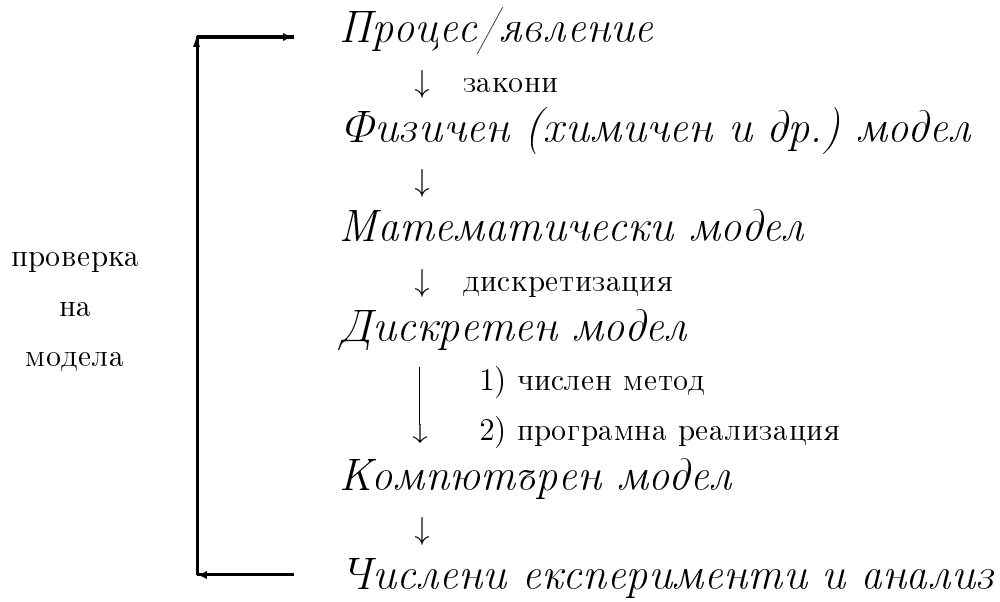
Обзор на изследваната област и актуалност на темата

Възникването на паралелните архитектури и развитието на компютърните технологии мотивира нарасналия интерес към конструирането на ефективни паралелни алгоритми за решаване на задачи от почти всички съвременни научни области. Настоящата дисертация също е насочена към конструиране и изследване на паралелни алгоритми.

Изучаването на дадено явление или процес изисква познаване на параметрите, които го описват (като концентрация, температура и др.). За целта е необходима процедура за количествена характеризация на параметрите. Директното измерване и наблюдение на интересуващите ни величини понякога се оказват твърде скъпи, а в много случаи са невъзможни. Тук на помощ идва *компютърното моделиране.* То е в основата на много от най-значимите достижения в съвременната наука и технологии. В области като молекулярна биология, ядрена физика, геофизика, метеорология, аеродинамика, електроника, материалознание и екология, компютърното моделиране е основно изследователско средство.

Компютърното моделиране е сложен интердисциплинарен процес, който обхваща последователност от различни модели, създадени на няколко стъпки. На първата стъпка, с помощта на подходящи закони и като се пренебрегнат по-несъществени фактори, се създава апроксимация на реалния процес – физичен (химичен и т. н.) модел. След това, той се преобразува в математически модел, адекватно отразяващ резултатите от планирани експерименти и наблюдения. Най-често математическият модел се състои от диференциални и/или интегрални уравнения, които описват неизвестните параметри на изследвания процес. В повечето случаи уравненията от математическия модел не могат да се решат в явен вид и затова се прилага числен метод за тяхната дискретизация. Така се създава дискретен модел, който обикновено се състои от системи линейни или нелинейни алгебрични уравнения. На този етап стават необходими: 1) ефективни методи и алгоритми за решаване на получе-

ните след дискретизацията системи алгебрични уравнения; 2) високопроизводителни компютърни програми, които в максимална степен използват възможностите и архитектурата на съвременните изчислителни системи. Конструирането на подходящ числен метод и написването на компютърна програма за неговата реализация (или изборът измежду вече съществуващите такива) представляват компютърен модел на изходния процес. Последният етап се състои от провеждане на изчислителни експерименти и визуализация и анализ на получените резултати. Целта е да се провери създадения модел и да се оцени адекватността му за изследване на изходното явление. Ако се налага подобряване на модела, се извършват отново част от стъпките на този процес. Основните етапи на компютърното моделиране са представени на



Фиг. 1: Етапи на компютърното моделиране.

Фигура 1. Независимо от огромния прогрес по отношение на производителността на изчислителната техника, определящи за развитието на компютърното моделиране са постиженията в областта на числените методи и алгоритмите за тяхната реализация.

Полученото чрез компютърно моделиране решение е толкова по-близо до реалното поведение на изследвания процес, колкото по-точни са конструираните на всяка стъпка модели. В същото време, по-голямата точност води до по-сложни модели, при което нараства нуждата от използването на съвременни паралелни компютри. Това налага създаването на подходящи дискретни модели и конструирането на паралелни алгоритми и програми за решаване на съответните системи. Темата на настоящата дисертация е мотивирана от това приоритетно направление на компютърното моделиране.

Изходен математически модел за изследванията в дисертационния труд е класът

на линейните елиптични гранични задачи от втори ред. Освен самостоятелно значение, развитието в областта на числените методи и алгоритми за линейни елиптични задачи има принос и при решаването на нелинейни диференциални уравнения. Причината е, че итерационните методи за нелинейни задачи се състоят от редици от спомагателни линейни задачи. Общата теория разработена в книгата на I. Faragó, J. Karátson [46] дава възможност за ефективно прилагане на бързи алгоритми за класове линейни задачи при числено решаване на нелинейни елиптични задачи с помощта на преобуславящи оператори. В частност, предложеният от O. Axelsson, I. Faragó, J. Karátson [12] алгоритъм, основаващ се на разделяне на областта на подобласти, дава възможност за конструиране на оптимални методи и алгоритми за моделиране на многослойни нелинейни среди.

Съществуват разнообразни подходи за конструиране на дискретен модел. Тук ще спрем вниманието си на онези от тях, използвани при изследванията в дисертацията. Методът на крайните разлики (МКР) и методът на крайните елементи (МКЕ) са основни средства за дискретизация на диференциални уравнения (вж. напр. [3, 5, 55]). Тяхното прилагане води до решаване на система линейни алгебрични уравнения (СЛАУ) с голяма размерност. Матриците на такива системи са с разредена структура, т. е. броят на ненулевите елементи е пропорционален на броя на неизвестните. Това е едно от най-важните свойства на тези системи и е в основата на конструирането на ефективни алгоритми за тяхното решаване. Понятието *голяма размерност* се променя с нарастване на производителността на изчислителната техника. Съвременното разбиране за задачи с голяма размерност е свързано с брой на неизвестните от порядък $10^6 - 10^9$ и дори 10^{12}. МКР и МКЕ са представители на мрежовите методи. При определени предположения матриците на получаваните линейни системи са с близки свойства и дори могат да съвпадат. Независимо от това, има класове задачи, при които се предпочита един от тях. Изследванията в дисертацията използват терминологията и на двата метода в зависимост от свойствата на изходната диференциална задача.

МКР е особено привлекателен за прости области като правоъгълници и когато са използвани равномерни мрежи. Матриците, получени от тези дискретизации, често са добре структурирани, като се състоят от няколко ненулеви диагонала. Матрицата на съответната СЛАУ за задачи с разделящи се променливи може да се представи като сума от тензорни произведения с матрици, получени чрез диференчна апроксимация на едномерни диференциални оператори. За по-сложни области и задачи със специфични особености на коефициентите, МКЕ има определени предимства по отношение на общността и алгоритмите за неговата реализация.

Много важни приложни задачи се моделират чрез гранични задачи, които имат силна хетерогенност и анизотропия на средата. Смесеният метод на крайните елементи е известен като най-добрият универсално приложим подход за дискретизация на такива задачи (вж. напр. F. Brezzi, M. Fortin [30]). Тази техника води до увеличаване на размерността на системата ЛАУ, чието решаване допълнително се усложнява от факта, че задачата е от тип седлова точка. D.N. Arnold, F. Brezzi [9] са показали, че апроксимацията чрез смесен МКЕ с елементи на Raviart-Thomas от най-нисък ред е еквивалентна на стандартната апроксимация с линейни неконформни елементи на Crouzeix-Raviart. По-нататък, такива зависимости са установени и изследвани за други варианти на смесения метод и съответни неконформни КЕ [8, 34]. Повече подробности относно триъгълни неконформни КЕ могат да се намерят например в [40, 45, 47]. Четириъгълни неконформни КЕ, основаващи се на завъртени мултилинейни функции на формата, са предложени като алгоритмично прост подход за устойчива апроксимация по МКЕ на задачата на Стокс от R. Rannacher, S. Turek [78]. P. Hansbo, M.G. Larson [56] са приложили тези елементи за получаване на равномерно приближение за почти несвиваема еластична задача. Това са някои от причините за специалния интерес към неконформни КЕ при числено решаване на диференциални уравнения, което е и предмет на част от настоящата дисертация.

Със създаването на дискретния модел естествено възниква въпросът „Как да решим получената СЛАУ?“ В дисертацията спираме вниманието си на: 1) елиптична гранична задача с разделящи се променливи, дискретизирана по МКР; 2) елиптична гранична задача дискретизирана с неконформни четириъгълни КЕ. Двата метода за дискретизация са от мрежов тип и водят до СЛАУ със симетрична, положително определена и разредена матрица, но с различно разположение на ненулевите елементи (ненулева структура). Специален интерес представлява случая на голяма размерност. Директното прилагане на подходите, разработени и изследвани за конформни КЕ води до известни трудности в случая на неконформни КЕ. Основните причини са две, а именно: 1) получените матрици на коравина имат специфична разредена структура, различна от регулярния блочно-лентов случай; 2) пространствата от функции КЕ, съответстващи например на вложени мрежи не са вложени (вж. R. Blaheta, S. Margenov, M. Neytcheva [27]). В дисертацията е предложен подход за преодоляване на някои от трудностите, породени от специфичната структура на матрицата на коравина за неконформни КЕ.

Какви възможности има за решаване на получените СЛАУ? Нека е дадена елиптична гранична задача в областта $\Omega \subset \mathbb{R}^2$ и нека за нейното числено решаване е приложен мрежов числен метод. При много общи предположения диференциална-

та задача е сведена до система линейни уравнения със симетрична, положително определена и разредена матрица A с размерност $N \times N$. Методите за решаване на получената СЛАУ се разделят на две големи групи – преки и итерационни. С възникването на векторните (в средата на 70-те) и паралелните (в средата на 80-те години на XX век) компютри (вж. J. Dongarra et al. [41]), всяка от тях се разделя на още две – говорим за последователни и паралелни алгоритми. В Таблица 1 са представе-

Таблица 1: Изчислителна сложност на методи за решаване на системи със симетрични и положително определени разредени матрици.

метод	изч. сл.
Метод на Холецки: LDL^t факторизация.	$\mathcal{O}\left(N^2\right)$
Метод на вложените сечения: ND.	$\mathcal{O}\left(N^{3/2}\right)$
Метод на спрегнатия градиент: CG.	$\mathcal{O}\left(N^{3/2}\right)$
Метод на спрегнатия градиент с преобуславяне: PCG. Преобуславяне по метода на непълната факторизация.	$\mathcal{O}\left(N^{5/4}\right)$
Метод на спрегнатия градиент с преобуславяне: PCG. Почти оптимален преобусловител.	$\mathcal{O}\left(N \log N\right)$
Метод на спрегнатия градиент с преобуславяне: PCG. Оптимален преобусловител.	$\mathcal{O}\left(N\right)$

ни данни за изчислителната сложност на някои от основните преки и итерационни методи за решаване на такъв клас линейни системи.

За умерени размерности получени при относително груба дискретизация на областта, успешно се прилагат преки алгоритми като метод на Холецки (LDL^t факторизация) и метод на вложените сечения (Nested Dissection). Методът на Холецки има изчислителна сложност $\mathcal{O}\left(N^2\right)$ при предположение, че се отчита лентовата структура или профила на матрицата при подходяща номерация на неизвестните. Методът на вложените сечения с изчислителна сложност $\mathcal{O}\left(N^{3/2}\right)$ е най-бързият измежду преките методи при общи предположения за областта и коефициентите на диференциалната задача. В основата на този метод е рекурсивното разделяне на графа, представящ структурата на ненулевите елементи на матрицата A.

С увеличаване размерността на линейната система все по-съществени стават преимуществата на итерационните методи. Едни от най-често прилаганите итерационни методи са от тип спрегнат градиент. Тяхната скорост на сходимост, а следователно и общата изчислителна сложност, зависи от числото на обусловеност на съответната матрица. Методът на спрегнатия градиент има изчислителна сложност, която

асимптотично съвпада със сложността на най-добрия пряк метод. По тази причина и поради по-малките изисквания за памет е прието, че предимствата на итерационните методи за решаване на задачи с достатъчно голяма размерност са безспорни.

Ефективен метод за решаване на големи разредени СЛАУ е методът на спрегнатия градиент с преобуславяне (PCG). При него вместо система $A\mathbf{x} = \mathbf{f}$ се решава еквивалентна система $\mathcal{C}^{-1}A\mathbf{x} = \mathcal{C}^{-1}\mathbf{f}$, където $\mathcal{C}$ е т. нар. преобуславяща матрица за A (преобусловител). Матрицата $\mathcal{C}$ се избира така, че системи с $\mathcal{C}$ да се решават съществено по-бързо отколкото с A и числото на обусловеност на $\mathcal{C}^{-1}A$ да е значително по-малко от това на A. Ефективността на методите на спрегнатия градиент с преобуславяне се определя от качествата на преобусловителя. Известните подходи за конструиране на ефективни преобусловители се основават най-често на: а) поточкова и блочна непълна факторизация; б) методи за декомпозиция; в) влагане на областта в стандартна област.

Алгоритмите от тип поточкова непълна факторизация ILU (виж напр. [41, 50, 81]) са едни от най-често прилаганите, благодарение на тяхната алгоритмична простота и добра сходимост за задачи с локални особености. В случая на симетрични и положително определени матрици, модифицираната непълна факторизация от нулев ред се означава с MIC(0) (Modified Incomplete Cholesky). Този преобусловител се прилага, когато матрицата A е M-матрица, а изчислителната му сложност е $\mathcal{O}\left(N^{5/4}\right)$. Проблем при ILU преобусловителите е, че както факторизацията, така и решаването на системи с факторизираната матрица се разпаралелват трудно поради рекурсивния им характер. Разработването на варианти за блочна непълна факторизация BILU дава възможност за паралелна реализация. Използването на BILU обикновено не води до оптимални преобусловители, т. е. до методи с изчислителна сложност $\mathcal{O}(N)$, но с тяхна помощ се получават ефективни паралелни алгоритми (виж [26, 35, 36, 53, 54, 81]). Блочната циркулантна факторизация предложена и изследвана от I. Lirkov et al. [71, 72] е почти оптимален представител на класа блочни преобусловители (т.е. изчислителната сложност на получения алгоритъм е $\mathcal{O}(N \log N)$) и в същото време позволява ефективна паралелна реализация (I. Lirkov, S. Margenov, M. Paprzycki [70]). Общ недостатък на блочните методи е, че те не са достатъчно ефективни за задачи с локални особености. В дисертацията е предложена специална конструкция за СЛАУ получена чрез неконформни четириъгълни КЕ, с помощта на която се запазва изчислителната ефективност на поточковата непълна факторизация MIC(0) като едновременно с това се дава възможност за ефективна паралелна реализация.

При конструирането на методи за декомпозиция се използват свойствата на ди-

ференциалната задача и/или на МКЕ и съответните матрици на коравина. Един универсален подход е техниката за разделяне на областта на подобласти (Domain Decomposition (DD)). Основна характеристика на DD е свеждането на задачата в изходната област до задачи в припокриващи се или непripокриващи се подобласти [81, 84]. Широко приложение през последните години имат преобуславящи техники, при които DD се комбинира с конструиране на последователности от мрежи, включително с локално сгъстяване (вж. [28, 29, 38, 39, 44, 96]). С тяхна помощ се получават оптимални многомрежови (multigrid) и многонивови (multilevel) методи. При алгоритмите основаващи се на DD, задачите в подобластите могат да се решават независимо. По тази причина те се използват често за конструиране на ефективни паралелни преобусловители (вж. напр. [42, 84]).

Основната идея при подхода за влагане на областта (domain imbedding [77], fictitious domain) е изходната област да се вложи в по-голяма, като последната има проста геометрична структура (напр. правоъгълник или кръг). За повече подробности виж напр. дисертацията на T. Rossi [79] и цитираната в нея литература.

Тук специално споменахме техниките за разделяне на областта на подобласти и за влагане на областта по следната причина. И при двата подхода за конструиране на преобусловители, изходната задача може да се сведе до задачи в канонични области и/или с разделящи се променливи, а в настоящата дисертация конструираме ефективни алгоритми за тяхното решаване.

За елиптични задачи с разделящи се променливи върху правоъгълни мрежи са разработени високоефективни преки алгоритми. Те са известни като *бързи директни методи (fast direct methods, fast Poisson solvers, fast separable solvers)* и започват развитието си в средата на 60-те години на миналия век. При дискретизация чрез МКР по петточков шаблон „кръст“ или чрез на части линейни триъгълни крайни елементи, се получават блочно тридиагонални матрици. В такъв случай бързите директни методи могат да бъдат по-бързи от итерационните методи. Това ги прави особено подходящи при конструирането на техники за преобуславяне с малка изчислителна сложност за решаване на по-сложни задачи върху по-сложни области и мрежи, включително с локално сгъстяване. Последното е причина за продължителния интерес към този клас преки алгоритми и към разработването на техни паралелни версии.

Бързите директни методи се прилагат за конструиране на преобусловители за решаване на елиптични задачи от втори ред с плавно променящи се коефициенти, както и на задачи, при които може да се използва техниката за влагане на областта. При подхода за разделяне на областта на подобласти (например за уравнение на

Лаплас в L-образна или T-образна област), системата от ЛАУ с матрица за съставната мрежа може да се сведе до задачи с разделящи се променливи върху равномерни правоъгълни подобласти и върху интерфейса (границата) между тях, които се решават чрез бързи преки алгоритми. Решаването на елиптични задачи с разделящи се променливи има подобно приложение при задачи с локално сгъстяване на мрежата. Тук преобусловителят използва пряк метод върху регулярна груба мрежа, както и върху регулярна фина мрежа.

Алгоритмите за разделяне на променливите са бързи директни методи. Бързото преобразование на Фурие (FFT) предложено от J.W. Cooley, J.W. Tukey [37] е приложено за първи път от R. Hockney [60] за решаване на елиптични ЧДУ с константни коефициенти. FFT се използва за получаване на ефективни преки и итерационни алгоритми за по-широк клас задачи (вж. напр. [87, 33]). Основен недостатък на алгоритмите, основаващи се на FFT е предположението, че мрежата е равномерна поне в едно от координатните направления за двумерни задачи (поне в две направления за тримерни задачи). Въпреки това FFT продължава да е широко използван поради почти оптималната си изчислителна сложност $\mathcal{O}(N \log N)$ и факта, че съществуват множество ефективни паралелни реализации на FFT (вж. напр. [83, 88, 92, 73]).

Важен представител на алгоритмите за разделяне на променливите е т. нар. техника за непълно решаване на задачи с разредена дясна част (SRHS). Тя е предложена независимо от A. Banegas [13], W. Proskurowski [76] и Y. Kuznetsov, A.M. Matsokin [67]. SRHS успешно се прилага и за задачи върху неравномерни мрежи, включително и в тримерния случай. С помощта на SRHS може да се разработи нов клас бързи директни методи [65, 67, 93]. Също така SRHS може да се използва при конструиране на ефективни преобусловители например чрез техниката за влагане на областта, както е направено от T. Rossi [79].

Един широко използван бърз директен метод е алгоритъмът за циклична редукция FACR с изчислителна сложност $\mathcal{O}(N \log N)$. Процесът на циклична редукция CORF (cyclic odd-even reduction and factorization) предложен от R. Hockney [60, 61] е неустойчив, но съществуват устойчиви версии предложени от O. Buneman [32]. Методи за обобщена циклична редукция са описани от R. Sweet [89, 90] и P. Swarztrauber [86, 87]. Паралелни реализации на FACR за векторни и паралелни компютри са предложени и изследвани в [31, 62, 49, 91].

Бързият алгоритъм за разделяне на променливите (FASV) е близък до циклична редукция. Основава се на техниката на четно-нечетна елиминация в комбинация със SRHS. FASV е предложен от P. S. Vassilevski [93] и е изследван по-късно в дисертацията на T. Rossi [79]. Той е устойчив по конструкция тъй като техниката за

четно-нечетна факторизация е устойчива (вж. D. Heller [58]). Изчислителната сложност на FASV също е $\mathcal{O}(N \log N)$. Този пряк алгоритъм е използван в настоящия дисертационен труд при конструиране на преобусловител за решаване на анизотропни задачи дискретизирани с неконформни четириъгълни КЕ. Ефективно приложение на FASV при конструиране на преобусловители с техниката за разделяне на областта на подобласти за 3D задачи е показано от P. S. Vassilevski, S. Petrova [95]. Ефективни паралелни реализации на FASV са предложени в [75, 80], както и в настоящата дисертация.

Друг клас бързи директни методи са т. нар. марш алгоритми. Те са предложени от R. Bank и D. Rose [14, 15, 16], а по-късно са преформулирани от P. S. Vassilevski [94] с помощта на SRHS и FASV. Марш алгоритъмът (R. Bank, D. Rose [15]) има оптимална изчислителна сложност, но не е устойчив за големи стойности на N. По-късно е предложен обобщен вариант за задачи с постоянни коефициенти (R. Bank, D. Rose [16]), известен още като k-редукция, който в комбинация с FFT и циклична редукция води до полиномиално устойчив алгоритъм с изчислителна сложност $\mathcal{O}(N)$. Версията за задачи с променливи коефициенти (R. Bank [14]) е с изчислителна сложност $\mathcal{O}(N \log \frac{N}{k})$. В настоящата дисертация са предложени и изследвани паралелни реализации на обобщения марш алгоритъм за променливи коефициенти, както е формулиран от P. S. Vassilevski [94] и на негова модификация въведена от автора. До този момент на автора не са известни други работи касаещи паралелна реализация на марш алгоритми.

Цели на дисертационния труд

Основните цели при разработването на настоящата дисертация са в следните направления:

- Изследване и паралелна реализация на бързи алгоритми за решаване на елиптична гранична задача с разделящи се променливи. Теоретичен анализ на тяхната изчислителна сложност.
- Конструиране и изследване на ефективни преобусловители за последователно и паралелно итерационно решаване на елиптични гранични задачи дискретизирани с неконформни четириъгълни крайни елементи. Теоретичен анализ на тяхната изчислителна сложност.
- Експериментално изследване на последователните преки и итерационни алгоритми и сравнение на тяхната ефективност с получените теоретични оценки.

- Експериментално изследване на конструираните паралелни преки и итерационни алгоритми върху разпределени изчислителни системи с различни характеристики.

Методология на изследването

Основна мярка за ефективността на даден последователен алгоритъм е неговата изчислителна сложност, т. е. броят на аритметичните операции (ар. оп.) за реализацията му. При итерационните методи ефективността се определя от скоростта на сходимост и изчислителната сложност за една итерация. Скоростта на сходимост зависи от числото на обусловеност на матрицата на системата.

Ускорението S_P и ефективността E_P играят ключова роля в анализа на паралелните алгоритми. Те зависят от времената за изчисление и комуникации, които се представят най-общо с $T_a = \mathcal{N} * t_a$ и $T_{com} = c_1 * t_s + c_2 * t_w$. Тук $\mathcal{N}$ е броят на операциите, c_1 характеризира броя на етапите, на които са необходими комуникации, а c_2 е количеството обменени числа. Коефициентите c_1 и c_2 могат да бъдат константи или да зависят от броя на процесорите и/или размерността на задачата. Когато няма припокриване между изчисления и комуникации при реализацията на алгоритъма, общото време за изпълнение върху P процесора е $T_P = T_a(P) + T_{com}(P)$, а ускорението е $S_P = \dfrac{T_1}{T_P} = \dfrac{T_a(1)}{T_a(P) + T_{com}(P)}$. При изчисления, разпределени по равно между процесорите, ускорението зависи съществено от времето за комуникации – колкото по-малко е то, толкова по-добро ускорение има съответния алгоритъм. Параметрите t_a, t_s и t_w зависят от конкретния паралелен компютър. Най-голям измежду тях е t_s и той може да е стотици или дори хиляди пъти по-голям от t_w. По тази причина, един и същ паралелен алгоритъм може да има различно ускорение върху машини с различни характеристики.

Всички предложени и изследвани методи в представената дисертация имат ясна алгоритмична структура. Получените резултати имат изразено конструктивен характер. Методологията на изследване включва: а) оценка на изчислителната сложност; б) оценка на скоростта на сходимост на метода на спрегнатия градиент с преобуславяне; в) оценка на паралелната ефективност; г) програмна реализация, провеждане на числени експерименти и анализ на получените резултати.

В някои от доказателствата са преодолени съществени трудности от изчислителен характер. В тези случаи, в качеството на помощни средства, са използвани програмни системи за символни пресмятания.

При програмната реализация на конструираните и изследвани алгоритми са из-

ползвани езици за програмиране от високо ниво Fortran77 [74] и C [6, 7, 57]. Паралелните изчисления и комуникации са осъществени чрез библиотека за паралелно програмиране MPI [85].

Често използвани съкращения и означения

ЧДУ	–	частни диференциални уравнения;
СЛАУ	–	система(и) линейни алгебрични уравнения;
МКР	–	метод на крайните разлики;
МКЕ	–	метод на крайните елементи;
КЕ	–	краен елемент;
CG	–	метод на спрегнатия градиент;
PCG	–	метод на спрегнатия градиент с преобуславяне;
MP	–	поточков (mid-point) интерполационен оператор;
MV	–	интегрален (integral mid-value) интерполационен оператор;
MIC(0)	–	модифицирана непълна факторизация на Холецки;
(P)SRHS	–	(паралелен) алгоритъм за непълно решаване на задачи с разредена дясна част;
(P)FASV	–	(паралелен) бърз алгоритъм за разделяне на променливите;
(P)GMF	–	(паралелен) обобщен марш алгоритъм, използващ (P)FASV;
(P)GMS	–	(паралелен) обобщен марш алгоритъм, използващ (P)SRHS;
ар. оп.	–	аритметични операции;
$\mathcal{N}_{Alg}$	–	изчислителна сложност на алгоритъм Alg;
N	–	размерност на СЛАУ;
P, N_p	–	брой процесори;
$\kappa(A)$	–	спектрално число на обусловеност на матрицата A;
$\mathcal{C}$	–	преобусловител;
ε	–	коефициент на анизотропия;
t_a	–	време за изпълнение на една ар. оп. от един процесор;
t_s	–	време за инициализиране на комуникация;
t_w	–	време за изпращане на едно число до съседен процесор;
T_a	–	общо време за изчисления в даден алгоритъм;
T_{com}	–	общо време за комуникации в даден алгоритъм;
T_P	–	време за изпълнение на паралелен алгоритъм върху P процесора;
S_P, E_P	–	ускорение и ефективност върху P процесора;

Организация на изложението

Настоящата дисертация се състои от увод, четири глави и списък на цитираната литература.

Първа глава има въвеждащ характер. В нея са представени известни факти, използвани на различни етапи от изследванията в дисертацията. В първия раздел е формулирана изходната диференциална задача и е показано как се получават съответните СЛАУ. Разгледани са два подхода за дискретизация: 1) МКР при петточков шаблон „кръст“ за двумерна елиптична гранична задача с разделящи се променливи; 2) използване на четириъгълни неконформни КЕ за двумерна елиптична гранична задача в общ вид. В т. 1.2 са разгледани алгоритъма за решаване на системи чрез блочна факторизация и метода на спрегнатия градиент с преобуславяне. В последния трети раздел вниманието е насочено към въпроси, касаещи конструирането и оценяването на паралелни алгоритми. Описан е системен подход за конструиране, състоящ се от четири отделни стъпки: *разделяне, комуникация, агломерация* и *изобразяване.* В първите два етапа се набляга на въпроси независещи от машината, като паралелизъм и оптимална ефективност. В третия и четвъртия етап вниманието се насочва към конкретната паралелна архитектура и производителността. Въведени са понятията ускорение и ефективност като средства за оценяване качеството на паралелни алгоритми. Също така е представен модел на времената за изчисление T_a и комуникации T_{com} в зависимост от параметрите t_a, t_s, t_w и архитектурата на паралелната изчислителна система. Дадени са характеристиките на петте паралелни изчислителни системи, използвани при числените експерименти.

Основните научни приноси са представени в следващите три глави.

Втора глава е посветена на последователни бързи алгоритми за разделяне на променливите и приложението им при конструиране на преобусловители. В първия от четирите раздела на Глава 2 са описани стандартен (SV) и бърз (FASV) алгоритъм за разделяне на променливите и е сравнена теоретично тяхната изчислителна сложност. Представена е и същността на техниката за непълно решаване на задачи с разредена дясна част, която е съществена за изследваните преки алгоритми. Втори раздел е посветен на стандартен (SM) и обобщен (GMF) марш алгоритъм като отново е изследвана теоретично тяхната изчислителна сложност. Въведена е модификация GMS на обобщения марш алгоритъм, която има известни предимства по отношение на паралелните свойства.

В т. 2.3 са изследвани експериментално устойчивостта и изчислителната ефективност на методите от първите два раздела. Алгоритмите SV и FASV са устойчиви не-

зависимо от размерността на задачата. Въпреки оптималния брой ар. оп. необходими за изпълнение на SM, алгоритъмът е неприложим самостоятелно заради неустойчивостта си за задачи с голяма размерност. Алгоритмите GMF и GMS са асимптотично условно устойчиви, като се влияят от размерността на подзадачите решавани със SM. Сравнителният анализ на изчислителната сложност показва, че асимптотично GMF е най-бърз, следван от FASV, GMS и SV. При това съществуват интервали от размерности на задачата, за които поне един от „бавните“ алгоритми GMS и SV е по-бърз от „бързите“. Експерименталните данни показват, че с нарастване на размерността на задачата времената за изпълнение се увеличават в съответствие с теоретичната оценка за изчислителната сложност. Като цяло обаче времето за изпълнение на GMS е най-малко за по-широк диапазон размерности на задачата отколкото се очакваше от теоретичния анализ. Причината е, че освен изчислителната сложност, върху него влияят и фактори като структура на алгоритъма и средства на компилатора. Поради опростената си структура и предимствата си по отношение на паралелна реализация, GMS дава сериозни основания да се разглежда, заедно с другите бързи методи при конструирането на ефективни алгоритми (в частност и преобусловители) за решаването на по-сложни задачи.

В последния раздел е представено едно приложение на бързите алгоритми за разработване на ефективни преобусловители. Чрез локална модификация на елементните матрици на коравина е конструиран почти оптимален преобусловител $\mathcal{C}$ с разделящи се променливи за анизотропна елиптична задача от втори ред дискретизирана с неконформни четириъгълни крайни елементи. За решаването на получената СЛАУ е използван метод на спрегнатия градиент с преобуславяне. Системата с преобусловител $\mathcal{C}$ е решена чрез алгоритъм FASV. Използвана е стандартна мрежа за конструиране на $\mathcal{C}$ и наклонена мрежа за дискретизация на изходната задача. Получени са оценки за числото на обусловеност за двата варианта MP и MV на възловите базисни функции, които не зависят от размерността на задачата, а само от параметъра на анизотропия ε. Изчислителната сложност за една итерация е $\mathcal{O}(N \log N)$. Предложеният почти оптимален преобусловител е ефективен за големи задачи с не силно изменящи се коефициенти и слаба анизотропия.

В трета глава са предложени и изследвани паралелни версии PFASV, PGMF и PGMS на преките алгоритми FASV, GMF и GMS от Глава 2. До този момент на автора не са известни други работи отнасящи се за паралелна реализация на обобщения марш алгоритъм. Раздел 3.1 е посветен на PFASV, а т. 3.2 – на PGMF и PGMS. В тези два раздела е отговорено на въпросите: „Как се разпределят данни и изчисления?“ и „Какви комуникации са необходими?“ за всеки от предложените

алгоритми. За PFASV и PGMS са разработени две версии на паралелен SRHS, означени с PSRHSF и PSRHSG, всяка от които е описана при съответния алгоритъм. Раздел 3.3 е посветен на въпроса „Колко добри са конструираните алгоритми?“. Получени са теоретични оценки за паралелните времена и ускорение и е направено сравнение на тези оценки за трите алгоритъма върху три от най-често изследваните паралелни архитектури – пръстен, двумерна мрежа и хиперкуб. При всички алгоритми изчисленията са разпределени по равно между процесорите. Основното предимство на PGMS е, че етапите, на които се изискват комуникации са константа (общо 4) и не зависят от изпълнението на програмата (т.е. са статични). При PFASV (както и при PGMF) на част от стъпките (кои точно зависи от размерността на задачата N и броя на процесорите N_p) се извършват „глобални“ за група процесори комуникации. Големината на тази група и количеството данни отново зависят от N и N_p и се определят динамично, което отнема допълнително време и води до намаляване на ефективността. Въз основа на теоретичния анализ е направен изводът, че при системи с обща памет или с разпределена памет при малки стойности на t_s, t_w и $\frac{t_s}{t_w}$, алгоритмите PFASV и PGMF имат по-добра производителност. Асимптотично относно броя на процесорите, PGMS може да е по-добър за някои паралелни системи (напр. клъстери, мрежи от работни станции), както по отношение на реалното време за изпълнение, така и спрямо ускорението.

В последния раздел 3.4 на главата са представени резултати от числените експерименти проведени върху четири паралелни системи – Parmac, Grendel, Beowulf и Blue. Направеното сравнение на поведението на изследваните алгоритми, както помежду им, така и по отношение на машините потвърждава изводите направени в т. 3.3. Именно, последователният алгоритъм с асимптотично най-голяма изчислителна сложност в някои случаи има най-добро паралелно поведение и най-малко общо време за изпълнение.

Четвърта глава е посветена на паралелни итерационни алгоритми за реализиране на метода на спрегнатия градиент. Изследван е теоретично и експериментално нов паралелен преобусловител. С негова помощ се решават ефективно големи СЛАУ, получени след дискретизация на анизотропна двумерна елиптична гранична задача от втори ред чрез завъртени билинейни неконформни крайни елементи върху четириъгълници. В основата на предложения алгоритъм е модифицираната непълна факторизация на Холецки (Modified Incomplete Cholesky, MIC(0)) на разредени матрици. Тя е приложена върху локално конструирана апроксимация на изходната матрица на коравина, позволяваща устойчива MIC(0) факторизация. Полученият преобусловител има специална блочна структура, която запазва изчислителната ефективност

на поточковата непълна факторизация като едновременно с това дава възможност за паралелна реализация.

Същността на MIC(0) и на предложената поелементна конструкция са представени в раздел 4.1. Като резултат от локалния анализ в раздел 4.2 е получено, че: 1) разредената апроксимация B на матрицата на коравина A удовлетворява условията за устойчива MIC(0) факторизация и за двата случая MP и MV на възловите базисни функции; 2) матриците B и A са спектрално еквивалентни, като за относителното число на обусловеност са в сила равномерни оценки по отношение на размерността на задачата и на произволни скокове на коефициентите; 3) изчислителната сложност за една итерация на PCG алгоритъма е $\mathcal{O}(N)$. В раздел 4.3 е илюстрирана скоростта на сходимост на получения PCG алгоритъм в зависимост от параметъра на анизотропия. Експериментално се потвърждава, че $\kappa(\mathcal{C}^{-1}A) = \mathcal{O}(N^{\frac{1}{2}})$, където $\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)$, т. е. скоростта на сходимост е $\mathcal{O}(N^{\frac{1}{4}})$.

Раздел 4.4 е посветен на паралелната реализация на конструирания PCG алгоритъм. Разгледаните въпроси са свързани с разпределяне на данни и изчисления и необходими комуникации. Получена е теоретична оценка за паралелното време за изпълнение. Определящата част от необходимите комуникации са локални и производителността на алгоритъма слабо зависи от броя на процесорите и от архитектурата на паралелната система. Коефициентите пред t_s и t_w в оценката на времето за комуникации зависят от размерността на задачата. По тази причина производителността зависи съществено от стойностите на t_s и t_w. По-малките параметри на компютъра t_s и t_w намаляват времето за комуникации и водят в общия случай до по-добри ускорение и ефективност. В частност, при паралелен компютър с обща памет, на практика не се извършват комуникации. Това означава, че ускорението $S_{N_p} = \dfrac{T_1}{T_{N_p}}$ в този случай ще бъде близко до теоретичната си горна граница $S_{N_p} \leq N_p$. От получената оценка следва, че големи отношения t_s/t_w и t_s/t_a могат да доведат до намаляване на производителността с повече от 50% за големи дискретни задачи. Свойствата на паралелния алгоритъм са илюстрирани в т. 4.5 с числени експерименти върху два Beowulf клъстера. Получените резултати показват потенциала на разработения паралелен алгоритъм за реализации върху системи с обща памет. Резултатите представени в тази глава са приложени от G. Bencheva, I. Georgiev, S. Margenov [22] при конструирането на паралелен двунивов преобусловител за анизотропни задачи, дискретизирани чрез Crouzeix-Raviart триъгълни неконформни елементи.

Списък публикации по дисертацията

Основните резултати по дисертацията са публикувани в:

1. G. Bencheva, Comparative performance analysis of 2D separable elliptic solvers, *Proceedings of the XIII-th summer school „Software and Algorithms of Numerical Mathematics“*, Nečtiny, Czech Republic, (1999), 11–27.

2. G. Bencheva, Comparative analysis of marching algorithms for separable elliptic problems, *Numerical Analysis and its Applications, Springer LNCS,* **1988** (2001), 76–83.

3. G. Bencheva, MPI parallel implementation of a fast separable solver, *Large-Scale Scientific Computing, Springer LNCS,* **2179** (2001), 454–461.

4. G. Bencheva, Parallel implementation of a generalized marching algorithm, Technical Report ISC-04-10-MATH (2004) (`http://www.isc.tamu.edu/iscpubs/0410.ps`).

5. G. Bencheva, Parallel performance comparison of three direct separable elliptic solvers, *Large-Scale Scientific Computing, Springer LNCS,* **2907** (2004), 421–428, (Extended version: Technical Report ISC-03-02-MATH (`http://www.isc.tamu.edu/iscpubs/0302.ps`).)

6. G. Bencheva, S. Margenov, On a preconditioning strategy for rotated linear FEM elliptic systems, *Proceedings, PRISM'01*, University of Nijmegen, The Netherlands, (2001), 87–90.

7. G. Bencheva, S. Margenov, Parallel incomplete factorization preconditioning of rotated linear FEM systems, *Journal of Computational and Applied Mechanics,* **4(2)** (2003), 105–117.

8. G. Bencheva, S. Margenov, Performance analysis of a parallel MIC(0) preconditioning of rotated bilinear nonconforming FEM systems, *Mathematica Balkanica,* **17** (2003), 319–335.

Три от представените публикации по дисертацията са в Lecture Notes in Computer Science – поредица с „импакт-фактор“ (impact factor 0.872 за 1999г.).

Апробация на резултатите

Резултатите включени в дисертацията са докладвани на семинара на секции „Паралелни алгоритми“ и „Научни пресмятания“ на Институт по паралелна обработка

на информацията (ИПОИ) при БАН. Част от тях са представени и на семинари на групите по Числени методи в Texas A&M University, College Station, Texas, USA и в Institute of Geonics, Academy of Sciences of Czech Republic, Ostrava, Czech Republic.

Съществени части от дисертацията са представени на следните специализирани научни конференции: XIII-th summer school „Software and Algorithms of Numerical Mathematics“, Nečtiny, Czech Republic, September 1999; Second Conference on Numerical Analysis and Applications, Rousse, Bulgaria, June 2000; International Workshop on Parallel Matrix Algorithms and Applications, Neuchâtel, Switzerland, August 2000; International Conference on Preconditioned Robust Iterative Solution Methods for Problems with Singularities, University of Nijmegen, The Netherlands, May 2001; 3rd International Conference on „Large-Scale Scientific Computations“, Sozopol, Bulgaria, June 2001; An Euro Conference on Numerical Methods and Computational Mechanics, University of Miskolc, Miskolc, Hungary, July 2002; Fifth International Conference on Numerical Methods and Algorithms, Borovets, Bulgaria, August 2002; Second Workshop on Parallel Matrix Algorithms and Applications, Neuchâtel, Switzerland, November 2002; 4th International Conference on „Large-Scale Scientific Computations“, Sozopol, Bulgaria, June 2003.

Основни научни приноси

Основните резултати, получени в дисертацията, имат конструктивен характер. Те са:

1. Направен е теоретичен и експериментален сравнителен анализ на пет алгоритъма за разделяне на променливите – SV, FASV, SM, GMF, GMS. Предложената в дисертацията оригинална модификация GMS на GMF опростява структурата на обобщения марш алгоритъм и има предимства по отношение на паралелната реализация.

2. Конструиран е и е изследван теоретично и експериментално преобусловител с разделящи се променливи за анизотропни елиптични гранични задачи, дискретизирани с неконформни четириъгълни КЕ. Получените оценки за числото на обусловеност са равномерни относно размерността на задачата и възможни скокове на коефициентите. Получена е характеризация на влиянието на коефициента на анизотропия върху ефективността на преобусловителя.

3. Предложени са нови паралелни реализации на алгоритмите FASV, GMF, GMS и е направен теоретичен сравнителен анализ на техните свойства.

4. Предложен е нов паралелен преобусловител за анизотропни елиптични гранични задачи, дискретизирани с неконформни четириъгълни КЕ. Приложена е MIC(0) факторизация върху локално конструирана апроксимация B на матрицата на коравина A. За числото на обусловеност $\kappa(B^{-1}A)$ са получени равномерни оценки относно размерността на дискретната задача и скоковете на коефициентите.

5. Получена е характеризация на паралелната ефективност на конструираните преки и итерационни алгоритми за класове паралелни изчислителни системи с разпределена памет.

6. Направен е експериментален сравнителен анализ на поведението на всеки от предложените паралелни алгоритми върху разпределени системи с различни характеристики.

Благодарности

Изказвам сърдечна благодарност на научния си ръководител ст.н.с. д-р Панайот Василевски и на научния си консултант ст.н.с. I ст. дмн Светозар Маргенов за постоянното внимание и съдействие по разработваната тема, както и на колективите на секции „Научни пресмятания“ и „Паралелни алгоритми“ при ИПОИ-БАН за стимулиращата творческа атмосфера.

Също така благодаря на групите по Числени методи в University of Nijmegen, The Netherlands, Institute of Geonics, AS CR и Texas A&M University, TX USA за полезните дискусии при разработване на различни части от дисертацията, както и за възможността по време на научните си визити да използвам наличните при тях паралелни изчислителни системи.

Без да ги познавам лично, съм признателна и на авторите на книгите [6, 57, 59], които оказват неоценима помощ на всеки начинаещ учен в областта на изчислителната математика при разработването и тестването на програми [6, 57] и при подготовката на доклади и статии [59].

От сърце благодаря на семейството си за проявеното търпение и оказаната подкрепа в трудните моменти съпътстващи изследванията и документирането на резултатите представени в настоящата дисертация.

Глава 1

Предварителна информация

Настоящата глава има въвеждащ характер. Тук са представени известни факти, използвани на различни етапи от изследванията в дисертацията. Най-напред е формулирана изходната диференциална задача и е показано накратко как се получават съответните системи линейни алгебрични уравнения при два подхода за дискретизация. След това са разгледани метода за решаване на системи чрез блочна факторизация и алгоритъма на спрегнатия градиент с преобуславяне. В последната част вниманието е насочено към въпроси, касаещи конструирането и оценяването на паралелни алгоритми.

1.1 Постановка на задачата и дискретизация

Предмет на изследванията в настоящата дисертация са числени методи (преки и итерационни) за решаване на системи линейни алгебрични уравнения от вида $A\mathbf{x} = \mathbf{f}$ получени след дискретизация на елиптичната гранична задача от втори ред

$$
\left|\begin{array}{rcll}
-\nabla \cdot (a(y)\nabla u(y)) & = & f(y) & \text{в} \quad \Omega, \\
u & = & \mu(y) & \text{върху} \quad \Gamma_D, \\
(a(y)\nabla u(y)) \cdot n & = & g(y) & \text{върху} \quad \Gamma_N.
\end{array}\right. \tag{1.1.1}
$$

Тук $y = (y_1, y_2)$, с $\nabla u(y)$ е означен градиента на $u(y)$ ($\nabla u(y) = (\frac{\partial u}{\partial y_1}, \frac{\partial u}{\partial y_2})^T$), а $\nabla \cdot q$ е дивергенцията на q ($q = (q_1, q_2)^T$, $\nabla \cdot q = \frac{\partial q_1}{\partial y_1} + \frac{\partial q_2}{\partial y_2}$). По-нататък се предполага, че Ω е изпъкнала многоъгълна област в $\mathbb{R}^2$, $f(y)$, $\mu(y)$, $g(y)$ са дадени функции в $L^2(\Omega)$, n е единична външна нормала към границата $\Gamma = \partial\Omega$ и $\Gamma = \Gamma_D \cup \Gamma_N$, $meas(\Gamma_D) \neq 0$. Функциите $a_{ij}(y)$ са на части гладки върху $\overline{\Omega} = \Omega \cup \Gamma$ като матрицата $a(y) = \{a_{ij}(y)\}_{i,j=1}^2$ е симетрична и равномерно положително определена в Ω.

Структурата на получената СЛАУ зависи пряко от начина на дискретизация. Измежду най-известните и често използвани подходи за апроксимация са метода на крайните разлики (МКР) и метода на крайните елементи (МКЕ).

В следващите две секции е показано, как се получава линейната система за: а) елиптична гранична задача с разделящи се променливи, дискретизирана по МКР при петточков шаблон „кръст“; б) елиптична гранична задача, дискретизирана с помощта на четириъгълни неконформни крайни елементи. В следващите глави са конструирани и изследвани ефективни последователни и паралелни алгоритми за решаването на СЛАУ за тези два случая.

1.1.1 Метод на крайните разлики за задача с разделящи се променливи

Разглеждаме задача на Дирихле за елиптично уравнение от втори ред с разделящи се променливи от вида:

$$\left| \begin{array}{rcl} -\sum\limits_{s=1}^{2} \dfrac{\partial}{\partial y_s}\left(a_s(y_s)\,\dfrac{\partial u}{\partial y_s}\right) &=& f(y), \quad y \in \Omega \\ u &=& \mu(y), \quad y \in \partial\Omega \end{array} \right. , \tag{1.1.2}$$

където $y = (y_1, y_2)$ и $\Omega = (0,1)^2$, а функциите $a_s(y_s)$, $f(y)$ и $\mu(y)$ са достатъчно гладки.

Задачата (1.1.2) е частен случай на (1.1.1), при който $a(y)$ е диагонална матрица, a_{ii} зависят само от y_i и границата $\Gamma_N = \emptyset$. По-долу е описано как се получава съответната СЛАУ, като се приложи МКР при петточков шаблон „кръст“ върху равномерна мрежа (А.А. Самарский [3]). Същата дискретна задача се получава и чрез на части линейни триъгълни крайни елементи върху равномерна мрежа (Г. Стренг, Дж. Фикс [5]).

Използваме мрежа със стъпки $h_1 = \dfrac{1}{n+1}$, $h_2 = \dfrac{1}{m+1}$ ($m, n \in \mathbb{N}$) и вътрешни възли

$$(y_1^i, y_2^j) = (i\,h_1, j\,h_2),\ i = 1, \ldots, n,\ j = 1, \ldots, m. \tag{1.1.3}$$

Всеки от диференциалните оператори $\dfrac{\partial}{\partial y_s}\left(a_s(y_s)\,\dfrac{\partial u}{\partial y_s}\right)$, $s = 1, 2$ се апроксимира с диференчен оператор от вида

$$\frac{1}{h_s}\left(a_s(y_s + \frac{1}{2}h_s)\,u_{y_s} - a_s(y_s - \frac{1}{2}h_s)\,u_{\overline{y}_s}\right),\ s = 1, 2\,, \tag{1.1.4}$$

където

$$u_{y_1} = \frac{u(y_1+h_1,y_2)-u(y_1,y_2)}{h_1}\,,\; u_{y_2} = \frac{u(y_1,y_2+h_2)-u(y_1,y_2)}{h_2}\,,$$
$$u_{\overline{y}_1} = \frac{u(y_1,y_2)-u(y_1-h_1,y_2)}{h_1}\,,\; u_{\overline{y}_2} = \frac{u(y_1,y_2)-u(y_1,y_2-h_2)}{h_2}\,.$$

За апроксимация на дясната страна $f(y)$ се избира мрежова функция $\varphi(y)$ такава, че $\varphi(y)-f(y)=\mathcal{O}\left(|h|^2\right)$, където $|h|^2=h_1^2+h_2^2$. Понеже $f(y)$ е непрекъсната, може да се вземе $\varphi(y)=f(y)$ за $y\in\{(y_1^i,y_2^j), i=1,\ldots,n,\ j=1,\ldots,m\,\}$.

Като се въведе означението $\widetilde{f}_{i,j}=f(y_1^i,y_2^j)$, на диференциалната задача (1.1.2) се съпоставя диференчната задача на Дирихле

$$(1.1.5)\qquad \left|\begin{array}{l} -\dfrac{1}{h_1^2}\left(a_{1,i-\frac12}\,x_{i-1,j}-\left(a_{1,i+\frac12}+a_{1,i-\frac12}\right)x_{i,j}+a_{1,i+\frac12}\,x_{i+1,j}\right) \\ -\dfrac{1}{h_2^2}\left(a_{2,j-\frac12}\,x_{i,j-1}-\left(a_{2,j+\frac12}+a_{2,j-\frac12}\right)x_{i,j}+a_{2,j+\frac12}\,x_{i,j+1}\right)=\widetilde{f}_{i,j}\,, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i=1,\ldots,n, j=1,\ldots,m, \\ x_{0,j}=\mu(0,y_2^j),\quad x_{n+1,j}=\mu(1,y_2^j),\quad j=1,\ldots,m, \\ x_{i,0}=\mu(y_1^i,0),\quad x_{i,m+1}=\mu(y_1^i,1),\quad i=1,\ldots,n, \end{array}\right.\,.$$

Тук $x_{i,j}$ е стойността на търсената функция във възел (y_1^i,y_2^j) на мрежата (1.1.3).

Дефиниция 1.1.1 *Нека C е матрица с размерност $m_1\times n_1$ и елементи $c_{i,j}$, а D е матрица с размерност $m_2\times n_2$. Матрицата*

$$C\otimes D=\begin{pmatrix} c_{1,1}\,D & c_{1,2}\,D & \ldots & c_{1,n_1}\,D \\ c_{2,1}\,D & c_{2,2}\,D & \ldots & c_{2,n_1}\,D \\ \ldots & \ldots & \ldots & \ldots \\ c_{m_1,1}\,D & c_{m_1,2}\,D & \ldots & c_{m_1,n_1}\,D \end{pmatrix}$$

е с размерност $m_1\,m_2\times n_1\,n_2$ и се нарича тензорно произведение на матриците C и D.

Нека $T=\{t_{i,j}\}_{i,j=1}^{n}$ и $B=\{b_{i,j}\}_{i,j=1}^{m}$ са матричните представяния на диференчните оператори (1.1.4) съответно за $s=1$ и $s=2$. Те са тридиагонални, симетрични и положително (полу-)определени матрици и имат вида:

$$T=\frac{1}{h_1^2}\begin{pmatrix} a_{1,1+\frac12}+a_{1,1-\frac12} & -a_{1,1+\frac12} & 0 & 0 \\ -a_{1,2-\frac12} & a_{1,2+\frac12}+a_{1,2-\frac12} & -a_{1,2+\frac12} & 0 \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ 0 & -a_{1,n-1-\frac12} & a_{1,n-1+\frac12}+a_{1,n-1-\frac12} & a_{1,n-1+\frac12} \\ 0 & 0 & -a_{1,n-\frac12} & a_{1,n+\frac12}+a_{1,n-\frac12} \end{pmatrix},$$

$$B = \frac{1}{h_2^2}\begin{pmatrix} a_{2,1+\frac{1}{2}} + a_{2,1-\frac{1}{2}} & -a_{2,1+\frac{1}{2}} & 0 & 0 \\ -a_{2,2-\frac{1}{2}} & a_{2,2+\frac{1}{2}} + a_{2,2-\frac{1}{2}} & -a_{2,2+\frac{1}{2}} & 0 \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ 0 & -a_{2,m-1-\frac{1}{2}} & a_{2,m-1+\frac{1}{2}} + a_{2,m-1-\frac{1}{2}} & a_{2,m-1+\frac{1}{2}} \\ 0 & 0 & -a_{2,m-\frac{1}{2}} & a_{2,m+\frac{1}{2}} + a_{2,m-\frac{1}{2}} \end{pmatrix}.$$

Ако се използва подреждане на неизвестните по хоризонтални линии, т.е. по линии $y_2 = const$, системата (1.1.5) се представя чрез тензорно произведение на матрици по следния начин:

$$A\mathbf{x} = \mathbf{f}\,, \tag{1.1.6}$$

където

$$A = B \otimes I_n + I_m \otimes T$$

$$= \begin{pmatrix} T + b_{1,1}\, I_n & b_{1,2}\, I_n & 0 & \dots & 0 \\ b_{2,1}\, I_n & T + b_{2,2}\, I_n & b_{2,3}\, I_n & \dots & 0 \\ \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots \\ 0 & \dots & b_{m-1,m-2}\, I_n & T + b_{m-1,m-1}\, I_n & b_{m-1,m}\, I_n \\ 0 & \dots & 0 & b_{m,m-1}\, I_n & T + b_{m,m}\, I_n \end{pmatrix},$$

$$\mathbf{x}^T = (\mathbf{x}_1^T\,,\, \mathbf{x}_2^T\,,\, \dots\,,\, \mathbf{x}_m^T),\ \mathbf{f}^T = (\mathbf{f}_1^T\,,\, \mathbf{f}_2^T\,,\, \dots\,,\, \mathbf{f}_m^T),\ \mathbf{x}_j\,,\, \mathbf{f}_j \in \mathbb{R}^n, j = 1, \dots, m,$$

$$\mathbf{x}_j = (x_{1,j}\,,\, x_{2,j}\,,\, \dots\,,\, x_{n,j})^T,\ \mathbf{f}_j = (f_{1,j}\,,\, f_{2,j}\,,\, \dots\,,\, f_{n,j})^T,$$

а I_n и I_m са единични матрици от ред n и m. Компонентите $f_{i,j}$ на дясната част се получават от $\widetilde{f}_{i,j}$ след налагане на граничните условия както следва

$$f_{1,j} = \widetilde{f}_{1,j} + \frac{1}{h_1^2}\, a_{1,1-\frac{1}{2}}\, x_{0,j}, \quad f_{n,j} = \widetilde{f}_{n,j} + \frac{1}{h_1^2}\, a_{1,n+\frac{1}{2}}\, x_{n+1,j}, \quad j = 2, \dots, m-1,$$

$$f_{i,1} = \widetilde{f}_{i,1} + \frac{1}{h_2^2}\, a_{2,1-\frac{1}{2}}\, x_{i,0}, \quad f_{i,m} = \widetilde{f}_{i,m} + \frac{1}{h_2^2}\, a_{2,m+\frac{1}{2}}\, x_{i,m+1}, \quad i = 2, \dots, n-1,$$

$$\begin{aligned} f_{1,1} &= \widetilde{f}_{1,1} + \frac{1}{h_1^2}\, a_{1,1-\frac{1}{2}}\, x_{0,1} + \frac{1}{h_2^2}\, a_{2,1-\frac{1}{2}}\, x_{1,0}, \\ f_{1,m} &= \widetilde{f}_{1,m} + \frac{1}{h_1^2}\, a_{1,1-\frac{1}{2}}\, x_{0,m} + \frac{1}{h_2^2}\, a_{2,m+\frac{1}{2}}\, x_{1,m+1}, \\ f_{n,1} &= \widetilde{f}_{n,1} + \frac{1}{h_1^2}\, a_{1,n+\frac{1}{2}}\, x_{n+1,1} + \frac{1}{h_2^2}\, a_{2,1-\frac{1}{2}}\, x_{n,0}, \\ f_{n,m} &= \widetilde{f}_{n,m} + \frac{1}{h_1^2}\, a_{1,n+\frac{1}{2}}\, x_{n+1,m} + \frac{1}{h_2^2}\, a_{2,m+\frac{1}{2}}\, x_{n,m+1}, \end{aligned}$$

$$f_{i,j} = \widetilde{f}_{i,j},\ i = 2, \ldots, n-1,\ j = 2, \ldots, m-1.$$

За хомогенни гранични условия е в сила $f_{i,j} = \widetilde{f}_{i,j},\ i = 1, \ldots, n,\ j = 1, \ldots, m$.

Ако се използва вертикално лексикографско подреждане на неизвестните, т.е. по линии $y_1 = const$, системата (1.1.5) се записва във вида:

$$A'\mathbf{x}' = \mathbf{f}', \tag{1.1.7}$$

където

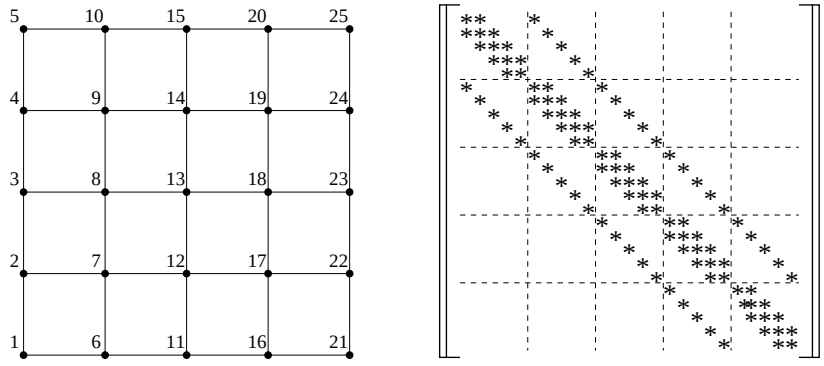
$$A' = T \otimes I_m + I_n \otimes B$$

$$= \begin{pmatrix} B + t_{1,1}\, I_m & t_{1,2}\, I_m & 0 & \ldots & 0 \\ t_{2,1}\, I_m & B + t_{2,2}\, I_m & t_{2,3}\, I_m & \ldots & 0 \\ \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots \\ 0 & \ldots & t_{n-1,n-2}\, I_m & B + t_{n-1,n-1}\, I_m & t_{n-1,n}\, I_m \\ 0 & \ldots & 0 & t_{n,n-1}\, I_m & B + t_{n,n}\, I_m \end{pmatrix},$$

$$\mathbf{x}'^T = (\mathbf{x}_1'^T, \mathbf{x}_2'^T, \ldots, \mathbf{x}_n'^T),\ \mathbf{f}'^T = (\mathbf{f}_1'^T, \mathbf{f}_2'^T, \ldots, \mathbf{f}_n'^T),\ \mathbf{x}_i', \mathbf{f}_i' \in \mathbb{R}^m, i = 1, \ldots, n,$$

$$\mathbf{x}_i' = (x_{i,1}, x_{i,2}, \ldots, x_{i,m})^T,\ \mathbf{f}_i' = (f_{i,1}, f_{i,2}, \ldots, f_{i,m})^T.$$

При стандартна мрежа и номерация на възлите (Фигура 1.1а)), ненулевата структура на матрицата е изобразена на Фигура 1.1б).



а) триангулация; б) матрица на коравина A

Фиг. 1.1: Номерация на възлите и ненулева структура на матрицата на коравина при стандартна триангулация на областта за МКР при петточков шаблон „кръст“.

В настоящата дисертация са предложени и изследвани преки методи за решаване на система (1.1.5). Съответните резултати са представени в Глава 2 и Глава 3.

1.1.2 Неконформни четириъгълни крайни елементи

Тук разглеждаме задачата (1.1.1) в случая на хомогенни гранични условия. Както е известно, това не води до ограничения на метода за решаване на системата, тъй като при нехомогенни условия матрицата на коравина не се променя. Вариационната формулировка на задачата гласи: *за дадено* $f \in L^2(\Omega)$ *да се намери функция* $u \in H^1_D(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ върху } \Gamma_D\}$, *удовлетворяваща уравнението*

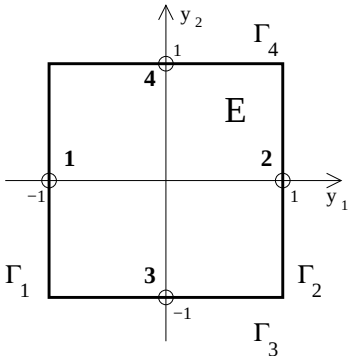
$$\mathcal{A}(u,v) = (f,v)\,, \qquad \forall v \in H^1_D(\Omega)\,, \tag{1.1.8}$$

където

$$\mathcal{A}(u,v) = \int_\Omega a(y)\nabla u(y)\cdot\nabla v(y)\,dy\,.$$

На непрекъснатата задача (1.1.8) се съпоставя дискретна по следния начин. Областта Ω се разделя на изпъкнали четириъгълници $e \in \omega_h$. Разделянето ω_h е съгласувано с точките на прекъсване на $a(y)$ така, че върху всеки елемент $e \in \omega_h$ функцията $a(y)$ е гладка. Освен това се предполага, че разделянето е квази-равномерно с характеристичен мрежов параметър h. Конструира се крайноелементно пространство V_h съответстващо на ω_h, като се използват завъртени билинейни неконформни крайни елементи (неконформни четириъгълни КЕ), както е предложено от R. Rannacher, S. Turek [78]. В дисертацията се разглеждат два алтернативни варианта на V_h (означени с MP и MV) в зависимост от избора на възлови базисни функции $\widehat{\phi}_i$.

Като базисен елемент E при дефинирането на изопараметричен ротиран билинеен елемент се използва единичен квадрат $(-1,1)^2$ със страни Γ_j, $j = 1,\ldots,4$ успоредни на координатните оси и възли $j = 1,\ldots,4$ в средите на страните му (виж Фигура 1.2). При дефинирането на възловите базисни функции $\widehat{\phi}_i \in S_p$, $S_p = span\{1, y_1, y_2, y_1^2 - y_2^2\}$



Фиг. 1.2: Базов неконформен краен елемент E.

за MP и MV се използват съответно поточков (mid-point) и интегрален (integral mid-value) интерполационен оператор. Стандартните интерполационни условия $\widehat{\phi}_i(j) =$

$\delta_{i,j}$, $i,j = 1,\ldots,4$ се използват при МР, където $\delta_{i,j}$ е символа на Кронекер. Условията за MV са

$$\frac{1}{|\Gamma_j|}\int_{\Gamma_j} \widehat{\phi}_i \, dy = \delta_{i,j},\ i,j = 1,\ldots,4\ ,$$

където Γ_j е страната на E съдържаща възел j. Изразите за $\widehat{\phi}_i$ в тези два случая са следните:

Базис МР	Базис MV
$\widehat{\phi}_1(y_1,y_2) = \frac{1}{4}(1-2y_1+(y_1^2-y_2^2))$	$\widehat{\phi}_1(y_1,y_2) = \frac{1}{8}(2-4y_1+3(y_1^2-y_2^2))$
$\widehat{\phi}_2(y_1,y_2) = \frac{1}{4}(1+2y_1+(y_1^2-y_2^2))$	$\widehat{\phi}_2(y_1,y_2) = \frac{1}{8}(2+4y_1+3(y_1^2-y_2^2))$
$\widehat{\phi}_3(y_1,y_2) = \frac{1}{4}(1-2y_2-(y_1^2-y_2^2))$	$\widehat{\phi}_3(y_1,y_2) = \frac{1}{8}(2-4y_2-3(y_1^2-y_2^2))$
$\widehat{\phi}_4(y_1,y_2) = \frac{1}{4}(1+2y_2-(y_1^2-y_2^2))$	$\widehat{\phi}_4(y_1,y_2) = \frac{1}{8}(2+4y_2-3(y_1^2-y_2^2))$

Нека означим с $\psi_e : E \to e$ съответната изопараметрична трансформация за всеки елемент $e \in \omega_h$. Тогава елементните възлови базисни функции се пресмятат по формулата

$$\{\phi_i\}_{i=1}^4 \ = \ \{\widehat{\phi}_i \circ \psi_e^{-1}\}_{i=1}^4 \, .$$

Те дефинират базис от прекъснати (неконформни) функции в пространството V_h.

Съответстващата на (1.1.8) дискретна задача получена по МКЕ е: *да се намери функция $u_h \in V_h$, удовлетворяваща уравнението*

$$\mathcal{A}_h(u_h, v_h) = (f, v_h)\,, \qquad \forall v_h \in V_h, \tag{1.1.9}$$

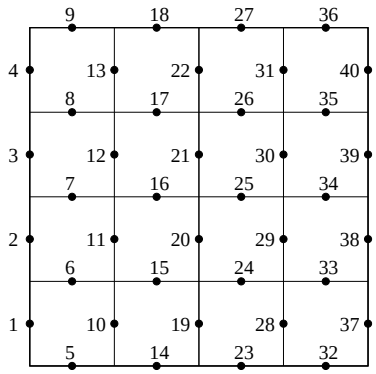
където

$$\mathcal{A}_h(u_h, v_h) = \sum_{e\in\omega_h} \int_e a(e)\nabla u_h \cdot \nabla v_h \, dy\,.$$

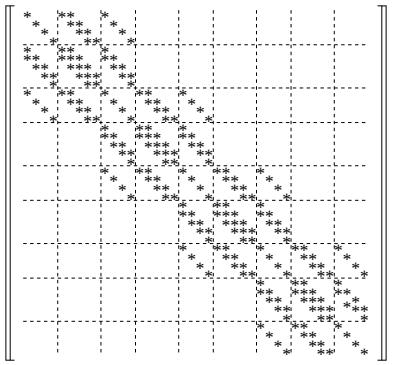
Тук $a(e)$ е дефинирана като интегрално усреднена стойност, т.е. $a(e) = \dfrac{1}{|e|}\displaystyle\int_e a(y)\,dy$ за всеки елемент $e \in \omega_h$. Съществено е, че за тази дискретизация се допускат големи скокове на коефициентите по границите между елементите. Задача (1.1.9) се свежда до линейната система уравнения

$$A\mathbf{x} = \mathbf{b}\,. \tag{1.1.10}$$

На практика това става чрез пресмятане и асемблиране на елементните матрици на коравина $A_e = \{\alpha_{ij}^e\}_{i,j=1}^4$, $\alpha_{ij}^e = \displaystyle\int_e a(e)\nabla\phi_i \cdot \nabla\phi_j \, dy$, $e \in \omega_h$, като се използва съответствие между локалната и глобалната номерация на възлите (виж напр. Y. Saad [81]). Матрицата на коравина $A = \{a_{ij}\}_{i,j=1}^N$ от (1.1.10) е разредена, симетрична и положително определена с не повече от 7 ненулеви елемента на ред. Структурата ѝ е една и съща за МР и MV, но зависи от триангулацията и номерацията на възлите. За



а) триангулация



б) матрица на коравина A

Фиг. 1.3: Номерация на възлите и структура на матрицата на коравина при стандартно разделяне на областта на неконформни четириъгълни КЕ.

стандартна мрежа и подреждане на неизвестните (Фигура 1.3 а)), структурата на A е изобразена на Фигура 1.3 б).

В т. 2.4 и Глава 4 са предложени и изследвани ефективни итерационни методи за решаване на СЛАУ получени чрез дискретизация на елиптичната гранична задача с неконформни четириъгълни КЕ.

1.2 Базови последователни алгоритми

Решаването на системи ЛАУ чрез блочна Гаусова елиминация е в основата на марш алгоритмите, изследвани в Глава 2 и Глава 3. Като базов алгоритъм за предложените в т. 2.4 и Глава 4 итерационни методи е използван алгоритъмът на спрегнатия градиент с преобуславяне (PCG). В тази секция се представя накратко същността на блочната Гаусова елиминация и PCG. Повече подробности за блочна Гаусова елиминация и свойствата на допълнението на Шур могат да бъдат намерени напр. в [11, 50], а за PCG – напр. в [11, 50, 81].

1.2.1 Блочна Гаусова елиминация

Разглеждаме система линейни алгебрични уравнения със симетрична и положително определена матрица от вида $A\mathbf{x} = \mathbf{f}$, където

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}, \mathbf{f} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}.$$

Векторите $\mathbf{x}_i$ и $\mathbf{f}_i$ са с размерности, съответстващи на блоковете $A_{i,i}$ за $i = 1, 2$.

Нека запишем матрицата в блочно факторизиран вид, т. е.

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} & 0 \\ A_{2,1} & S \end{pmatrix} \begin{pmatrix} I & A_{1,1}^{-1} A_{1,2} \\ 0 & I \end{pmatrix},$$

където $S = A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}$ е допълнението на Шур (Schur).

Трябва да се реши системата:

$$\begin{pmatrix} A_{1,1} & 0 \\ A_{2,1} & S \end{pmatrix} \begin{pmatrix} I & A_{1,1}^{-1} A_{1,2} \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}.$$

Полага се

$$\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \begin{pmatrix} I & A_{1,1}^{-1} A_{1,2} \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}.$$

Тогава

$$\left| \begin{array}{rcl} A_{1,1}\mathbf{y}_1 & = & \mathbf{f}_1 \\ A_{2,1}\mathbf{y}_1 + S\mathbf{y}_2 & = & \mathbf{f}_2 \end{array} \right. .$$

Тази система се решава относно $\mathbf{y}$ и се получава

$$\left| \begin{array}{rcl} \mathbf{y}_1 & = & A_{1,1}^{-1}\mathbf{f}_1 \\ \mathbf{y}_2 & = & S^{-1}(\mathbf{f}_2 - A_{2,1} A_{1,1}^{-1}\mathbf{f}_1) \end{array} \right. . \tag{1.2.11}$$

При намерени $\mathbf{y}_1$ и $\mathbf{y}_2$, $\mathbf{x}_1$ и $\mathbf{x}_2$ се намират последователно от

$$\left| \begin{array}{rcl} \mathbf{x}_1 + A_{1,1}^{-1}A_{1,2}\mathbf{x}_2 & = & \mathbf{y}_1 \\ \mathbf{x}_2 & = & \mathbf{y}_2 \end{array} \right. \Rightarrow \left| \begin{array}{rcl} \mathbf{x}_2 & = & \mathbf{y}_2 \\ \mathbf{x}_1 & = & \mathbf{y}_1 - A_{1,1}^{-1}A_{1,2}\mathbf{y}_2 \end{array} \right.$$

като в последната система $\mathbf{y}_1$ и $\mathbf{y}_2$ се заместват с равните им от (1.2.11). По този начин, решаването на изходната задача се свежда до системата:

$$\left| \begin{array}{rcl} A_{1,1}\mathbf{y}_1 & = & \mathbf{f}_1 \\ S\mathbf{x}_2 & = & \mathbf{f}_2 - A_{2,1}\mathbf{y}_1 \\ A_{1,1}\mathbf{x}_1 & = & \mathbf{f}_1 - A_{1,2}\mathbf{x}_2 \end{array} \right. . \tag{1.2.12}$$

1.2.2 Метод на спрегнатия градиент с преобуславяне

И при двата начина на дискретизация на задача (1.1.1), разгледани в т. 1.1, се получава система ЛАУ от вида

$$A\mathbf{x} = \mathbf{b}\,. \tag{1.2.13}$$

Матрицата A е симетрична и положително определена от ред $N = dim\, A$. Освен това A е разредена матрица, като броят на ненулевите елементи във всеки ред зависи от

метода на дискретизация, а структурата ѝ зависи допълнително и от номерацията на възлите. Един от най-добрите известни итерационни алгоритми за решаване на големи линейни системи със симетрични и положително определени матрици е методът на спрегнатия градиент (Conjugate Gradient (CG) method).

Скоростта на сходимост на CG зависи от числото на обусловеност на матрицата $\kappa(A)$: при намаляване на $\kappa(A)$ сходимостта е по-бърза. За съжаление, в случая на елиптични задачи от втори ред обикновено $\kappa(A) = \mathcal{O}(N) = \mathcal{O}(h^{-2})$, където N е броят на неизвестните, а h е стъпката на мрежата. В такива случаи, изходната система от ЛАУ се модифицира така, че числото на обусловеност на новата задача

$$\mathcal{C}^{-1}A\mathbf{x} = \mathcal{C}^{-1}\mathbf{b} \tag{1.2.14}$$

да бъде съществено намалено. Този подход е известен като метод на спрегнатия градиент с преобуславяне (Preconditioned Conjugate Gradient (PCG) method).

Дефиниция 1.2.1 *Симетричната и положително определена матрица $\mathcal{C}$, участваща в (1.2.14) и* Алгоритъм PCG, *се нарича преобуславяща матрица или по-кратко преобусловител.*

Стандартният метод на спрегнатия градиент с преобуславяне за решаване на (1.2.13) с относителна грешка ϵ и преобусловител $\mathcal{C}$ може да бъде записан във вида:

Алгоритъм PCG

начално приближение	$\mathbf{x^0} = \mathcal{C}^{-1}\mathbf{b}$
начален резидуал	$\mathbf{r^0} = \mathbf{b} - A\mathbf{x^0}$
начален квазирезидуал	$\mathbf{h^0} = \mathcal{C}^{-1}\mathbf{r^0}$
начална посока на търсене	$\mathbf{d^0} = \mathbf{h^0}$, $\rho_0 = \mathbf{r^{0}}^t\,\mathbf{h^0}$
	for $k = 0, 1, 2, \ldots$
изчисли	$\alpha_k = \dfrac{\rho_k}{\mathbf{d^k}^t A\mathbf{d^k}}$
следващо приближение	$\mathbf{x^{k+1}} = \mathbf{x^k} + \alpha_k\mathbf{d^k}$
следващ резидуал	$\mathbf{r^{k+1}} = \mathbf{r^k} - \alpha_k A\mathbf{d^k}$
следващ квазирезидуал	$\mathbf{h^{k+1}} = \mathcal{C}^{-1}\mathbf{r^{k+1}}$
	$\rho_{k+1} = \mathbf{r^{k+1}}^t\mathbf{h^{k+1}}$
стоп критерий	if $\rho_{k+1} \leq \epsilon^2\,\rho_0$ then stop
изчисли	$\beta_{k+1} = \dfrac{\rho_{k+1}}{\rho_k}$
следваща посока на търсене	$\mathbf{d^{k+1}} = \mathbf{h^{k+1}} + \beta_k\mathbf{d^k}$
	end {цикъл по k}

Край {PCG}.

Алгоритъм CG се получава като частен случай на PCG при $\mathcal{C} = I$. Така описаният Алгоритъм PCG на всяка стъпка от итерационния процес включва едно решаване на система с преобуславящата матрица $\mathcal{C}$, едно умножение на матрицата A по вектор плюс две скаларни произведения и три векторни операции от типа умножение на скалар по вектор плюс вектор. От тук, за изчислителната сложност на една итерация се получава оценката

$$\mathcal{N}_{it}^{PCG}(A^{-1}\mathbf{b}) \approx \mathcal{N}(\mathcal{C}^{-1}\mathbf{r^{k+1}}) + \mathcal{N}(A\mathbf{d^k}) + 10N\,. \tag{1.2.15}$$

Горна граница за броя на итерациите се дава от следната теорема (виж напр. O. Axelsson [11] и C. Johnson [63]).

Теорема 1.2.1 *Нека с $N_{it}(\epsilon)$, $\epsilon > 0$ е означен броя на PCG итерациите, достатъчни за получаване на относителна грешка $\epsilon_k = \dfrac{\|r^k\|_{A^{-1}}}{\|r^0\|_{A^{-1}}} \leq \epsilon$. Тогава*

$$N_{it}(\epsilon) \leq \left\lceil \frac{1}{2}\sqrt{\kappa(\mathcal{C}^{-1}A)}\ln\frac{2}{\epsilon} \right\rceil . \tag{1.2.16}$$

Следователно за числения решаване на системата са достатъчни общо $\mathcal{O}(N_{it}(\epsilon)\mathcal{N}_{it}^{PCG}(A^{-1}\mathbf{b}))$ операции.

Изчисляването на $\|r^k\|_{A^{-1}}$ е свързано с обръщане на матрицата A и обикновено критерият за спиране на итерациите при реализиране на метода и при изследване на скоростта му на сходимост е достигане на относителна грешка $\epsilon_k = \dfrac{\|r^k\|_{\mathcal{C}^{-1}}}{\|r^0\|_{\mathcal{C}^{-1}}} \leq \epsilon$.

От оценката за изчислителната сложност на Алгоритъм PCG и от **Теорема 1.2.1** следва общата стратегия за конструиране на ефективни преобусловители. Целта е матрицата $\mathcal{C}$ да удовлетворява следните две условия, които в общия случай са в конфликт:

- Съществува ефективен алгоритъм за решаване на системи с преобусловителя $\mathcal{C}$. Това означава, че за изчислителната сложност трябва да е в сила условието $\mathcal{N}(\mathcal{C}^{-1}x) \ll \mathcal{N}(A^{-1}x)$.

- Относителното число на обусловеност е съществено по-малко от числото на обусловеност на изходната матрица, т.е. $\kappa(\mathcal{C}^{-1}A) \ll \kappa(A)$.

Дефиниция 1.2.2 *Казва се, че преобусловителят $\mathcal{C}$ е оптимален, когато $\mathcal{N}(\mathcal{C}^{-1}x) = \mathcal{O}(N)$, $\kappa(\mathcal{C}^{-1}A) = \mathcal{O}(1)$.*

В представената дисертация са предложени два типа преобусловители за решаване на елиптична гранична задача, дискретизирана с неконформни четириъгълни КЕ.

Първият (виж т. 2.4) е с разделящи се променливи върху специално конструирана мрежа. Вторият преобусловител е от тип поточкова непълна факторизация ILU (за повече подробности виж [50, 41]). В случая на симетрични положително определени матрици методът на Гаус се модифицира като се получава метода на Холецки. Съответната непълна факторизация от нулев ред се означава с MIC(0). Класическите ILU преобусловители са едни от най-често прилаганите, благодарение на тяхната алгоритмична простота и добра сходимост за задачи с локални особености. Проблем при ILU преобусловителите е, че както факторизацията, така и решаването на системи с факторизираната матрица се разпаралелват трудно поради рекурсивния им характер. С помощта на специална конструкция, предложена в Глава 4, този проблем е преодолян за съответния MIC(0) преобусловител.

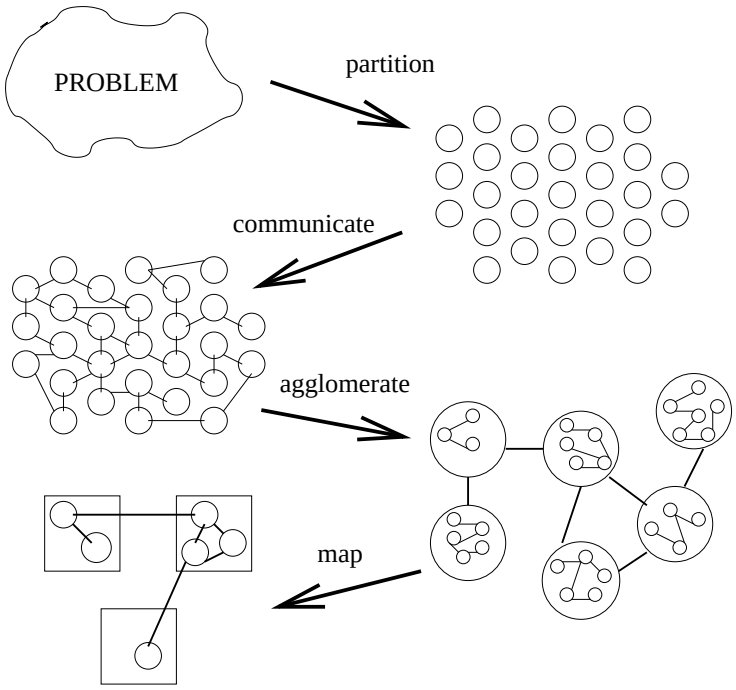
1.3 Паралелни алгоритми

Възникването на паралелните архитектури и прогресът в компютърните технологии са причина за нарасналия интерес в областта на паралелните алгоритми. През последните десетилетия се появиха много публикации, касаещи конструирането на паралелни версии на съществуващи преки и итерационни методи за числено решаване на ЧДУ. Основната задача на настоящата дисертация също е в тази област. Глави 3 и 4 са посветени на конструиране и изследване съответно на преки и итерационни паралелни алгоритми. В тази секция се разглеждат някои принципни въпроси свързани с разработването, реализацията и оценката на ефективността на паралелни алгоритми.

1.3.1 Общ подход за конструиране на паралелни алгоритми

Проектирането на ефективни паралелни алгоритми изисква комплексен анализ на конкретната задача и е трудно да се сведе до прости и универсални рецепти. Повечето задачи за програмиране имат няколко паралелни решения, като най-доброто може да се различава от това, предложено на базата на съществуващи последователни алгоритми.

В книгата на I. Foster [48] е описан полезен системен подход, който разглежда максимален обхват от възможности за избор, осигурява механизми за оценяване на алтернативи и редуцира броя на връщанията на предишни етапи, причинени от неподходящ избор. При тази методология означена накратко с PCAM, процесът на конструиране на паралелен алгоритъм се състои от четири отделни стъпки: *разделяне* (partitioning), *комуникация* (communication), *агломерация* (agglomeration) и



Фиг. 1.4: PCAM-схема за конструиране на паралелни алгоритми.

изобразяване (mapping). В първите два етапа се набляга на въпроси независещи от машината, като паралелизъм и оптимална ефективност и се търсят алгоритми с тези характеристики. В третия и четвъртия етап вниманието се насочва към конкретната паралелна архитектура и производителността. Тези четири етапа са илюстрирани на Фигура 1.4 и могат да се обобщят по следния начин:

1. *Разделяне.* Изчисленията, които трябва да се извършат и данните, с които се оперира се разделят на малки задания (по възможност – непресичащи се). Практически въпроси, като брой процесори в достъпния компютър се игнорират и вниманието се насочва върху откриване на възможности за паралелно изпълнение. Има два допълващи се подхода за генериране на разделяне, които могат едновременно да се прилагат към дадена задача и/или към различни нейни компоненти, за да се получат алтернативни паралелни алгоритми. При единия първо се определя подходящо разделяне на данните на задачата и след това се свързват изчисленията с данните. Тази техника се нарича *разделяне на областта* (domain decomposition). При алтернативния подход, известен като *функционално разделяне* (functional decomposition), първо се разделят изчисленията и след това се разглеждат данните.

2. *Комуникации.* Определят се необходимите комуникации за съгласуване на изпълнението на заданията и се дефинират подходящи комуникационни струк-

тури. Целта е да се разпределят операциите за комуникация (без да се въвеждат ненужни такива) върху много задания и да се организират по начин, който позволява едновременно изпълнение. Комуникационните структури се категоризират в четири двойки взаимно противоположни направления: а) *локални/глобални* – всяко задание комуникира с малко/голямо множество други задания; б) *структурирани/неструктурирани* – задание и неговите съседи образуват регулярна структура (напр. дърво или мрежа)/произволен граф; в) *статични/динамични* – вида на комуникационните структури не се променя с времето или се определя според изчислените данни на отделни етапи от изпълнението на алгоритъма; г) *синхронни/асинхронни* – производителят и потребителят на данните си съдействат/не си съдействат в операциите за обмен.

3. *Агломерация.* Заданията и комуникационните структури от първите два етапа се оценяват по отношение на конкретната производителност и реализацията. Целта е да се получи ефективен алгоритъм за определен клас паралелни компютри. Ако е необходимо, заданията се групират в по-големи и се дублират данни и/или изчисления, за да се подобри производителността или за да се намали цената за разработване.

4. *Изобразяване.* Всяко задание се свързва с процесор по такъв начин, че да се удовлетворят конкуриращите се условия за максимално използване на процесорите и минимизиране на комуникациите. Изобразяването може да бъде зададено статично или да се определя по време на изпълнението чрез алгоритми за балансиране на натоварването. (Върху системи с обща памет това се прави автоматично.)

Този подход може да се използва, независимо дали целта е да се създаде програма, която динамично генерира задания или трябва да има точно по едно задание за всеки процесор. Във втория случай (точно такива са и предложените в дисертацията паралелни алгоритми) въпросите свързани с изобразяването са включени в етапа на агломерация.

1.3.2 Оценка. Модел за времената за изчисление и комуникации.

Основна мярка за ефективността на даден последователен пряк алгоритъм е неговата изчислителна сложност. При итерационните методи това е скоростта на сходимост

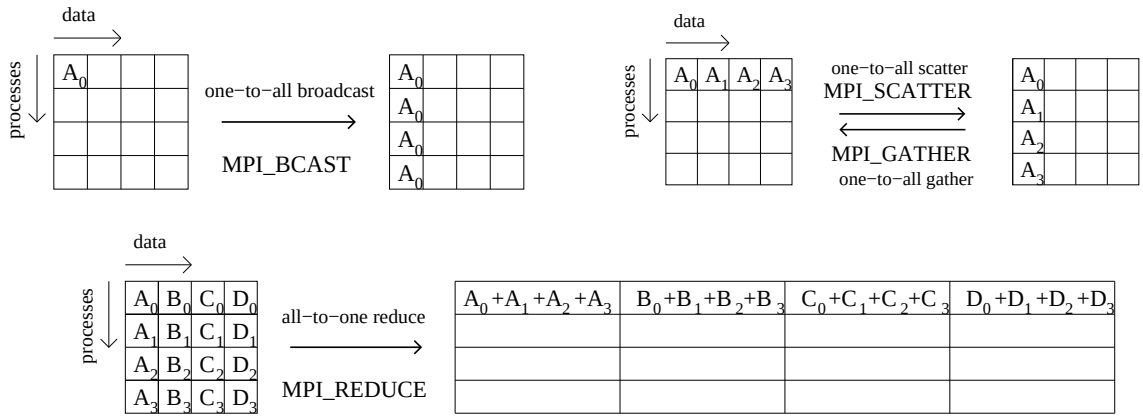
заедно с изчислителната сложност за една итерация. Ускорението S_P и ефективността E_P играят ключова роля в анализа на паралелните (преки и итерационни) алгоритми. Те се дефинират чрез

$$S_P = \frac{T_1}{T_P}\,, \quad E_P = \frac{S_P}{P} = \frac{T_1}{P.T_P}\,, \tag{1.3.17}$$

където T_1 и T_P са времената за изпълнение на алгоритъма съответно върху 1 и P процесора. Изразите (1.3.17) се наричат *относителни* ускорение и ефективност, защото са дефинирани по отношение на времето за изпълнение на паралелния алгоритъм върху един процесор. Теоретичните им горни граници са $S_P \leq P\,, E_P \leq 1$. При сравняване на два алгоритъма понякога се използват и т. нар. *абсолютни* ускорение и ефективност, които се основават на времето за изпълнение на най-добрия последователен алгоритъм за решаване на поставената задача върху един процесор.

Как може да се оцени времето T_P? Разбира се, то зависи освен от количеството и структурата на изчисления и комуникации, също и от характеристиките на конкретната машина и компилатора. Различни модели за оценка на T_P са представени в [48, 64, 82]. Анализът на паралелната изчислителна сложност на алгоритмите в Глави 3 и 4 е направен при формулираните по-долу предположения. Разглежда се паралелна система с разпределена памет от тип MIMD (Multiple Instruction Multiple Data) и P процесора. Използва се описание на архитектурата в термините на граф с върхове, съответстващи на процесорите и ребра, съответстващи на връзките между тях. Под разстояние между два процесора се разбира разстоянието между възлите в този граф. Предполага се, че изчисленията и комуникациите не се извършват едновременно и следователно времето за изпълнение на даден алгоритъм е сума от двете времена $T_P = T_a + T_{com}$. Приема се също, че няма векторизация и следователно времето за изпълнение на M аритметични операции (ар. оп.) върху един процесор се получава по формулата $T_a = M * t_a$, където t_a е усредненото време за извършване на една ар. оп. от един процесор. За оценка на времето за предаване на M числа между два процесора (one_to_one) на разстояние l един от друг, се използва формулата $oo(P,\, M) = t_s + l\, M * t_w$, където t_s е времето за стартиране на комуникацията, а t_w е времето, необходимо за изпращане на едно число до съседен процесор. Когато процесорите са съседни, т.е. $l = 1$, се говори за локална комуникация и времето е $T_{loc} = t_s + M * t_w$. В предложените паралелни алгоритми се използват и следните глобални комуникации (вж. Фигура 1.5):

- Предаване на M числа от един процесор до всички останали за време $b(P,\, M)$ (one_to_all broadcast);



Фиг. 1.5: Глобални комуникации.

- Един от процесорите изпраща до всеки от останалите по един пакет с данни, съдържащ M числа, за време $s(P, M)$ (one_to_all scatter). В общия случай данните в пакетите за отделните процесори са различни. Дуалната операция (all_to_one gather) – събиране на P такива пакета от всички останали в един процесор изисква същото време $s(P, M)$;

- Всеки процесор има пакет от M числа (в общия случай различен от пакетите на останалите процесори). След приключване на операцията (за време $r(P, M)$) един от процесорите притежава пакет, който е сума, разлика, максимум или друга операция приложена поелементно върху изходните пакети (all_to_one reduce). Схемата за извършването ѝ е обратна на one_to_all broadcast и следователно е в сила $r(P, M) = b(P, M)$.

Глобалните комуникации, както и тези между два несъседни процесора, зависят от паралелната архитектура и от общия брой процесори в нея. В дисертацията се използват горните граници за времената $oo(P, M)$, $b(P, M)$, $s(P, M)$ (виж Таблица 1.1),

Таблица 1.1: Времена за базовите комуникации.

	пръстен	квадратна мрежа	хиперкуб
$oo(P, M)$	$t_s + t_w.M.\lfloor P/2 \rfloor$	$t_s + 2t_w.M.\lfloor \sqrt{P}/2 \rfloor$	$t_s + t_w.M.\log P$
$b(P, M)$	$(t_s + t_w.M)\lceil P/2 \rceil$	$2(t_s + t_w.M)\lceil \sqrt{P}/2 \rceil$	$(t_s + t_w.M)\log P$
$s(P, M)$	$(t_s + t_w.M)(P-1)$	$2t_s(\sqrt{P}-1)+$	$t_s \log P+$
		$t_w.M.(P-1)$	$t_w.M.(P-1)$

получени във V. Kumar et al. [64] за три от най-често изследваните архитектури – пръстен, квадратна мрежа и хиперкуб.

Времената за изчисление и комуникации за даден алгоритъм се представят най-общо с $T_a = \mathcal{N} * t_a$ и $T_{com} = c_1 * t_s + c_2 * t_w$. Тук $\mathcal{N}$ е броят на операциите, c_1 характеризира броя на етапите, на които са необходими комуникации, а c_2 е количеството прехвърлени числа. Коефициентите c_1 и c_2 могат да бъдат константи или функции на броя на процесорите и/или размерността на задачата. Параметрите t_a, t_s и t_w зависят от паралелния компютър. Най-голям измежду тях е t_s и той може да е стотици или дори хиляди пъти по-голям от t_w. По тази причина, един и същ паралелен алгоритъм може да има различно ускорение върху машини с различни характеристики. Таблици със сравнения на t_s и t_w за няколко различни паралелни системи са дадени напр. в [42, 48].

1.3.3 Реализация – използван софтуер и хардуер

Съществуват разнообразни програмни средства (паралелни библиотеки, разширения на операционни системи и на езици за програмиране от високо ниво), които поддържат различни нива на паралелизъм (виж напр. [41, 42, 48]). За разработването на програми предназначени за MIMD компютри, най-популярна е библиотеката за паралелно програмиране MPI (Message Passing Interface) [85].

MPI е преносим стандарт за предаване на съобщения, включващ реализации на Fortran77 и C/C++. Той е достъпен върху различни софтуерни платформи (вкл. UNIX, LINUX, и някои версии на Windows). При създаване на MPI е направен опит за включване на всичко най-добро от по-ранни среди като PVM, Express, P4 и др. MPI осигурява модел, който скрива разликите в компютърните архитектури. Това позволява програмите, написани за компютър с разпределена памет да бъдат изпълнени успешно и върху система с обща памет, мрежа от работни станции, дори и върху хетерогенни системи, т. е. групи от процесори с различни архитектури.

Стандартът дефинира синтаксис и семантика на ядрото на 129 библиотечни функции, голяма част от които имат многобройни параметри или варианти. С тяхна помощ се поддържат: комуникации между два процесора; групови операции; групи от процеси; комуникационни области; топологии от процеси. MPI не дефинира: явни операции с обща памет; средства за конструиране на програми; средства за откриване на грешки; входно/изходни функции. По-задълбочена представа за възможностите на MPI и начина на използване на функциите му, може да се получи напр. от [42, 48].

Алгоритмите в тази дисертация са реализирани на C, като за осъществяване на паралелните изчисления и комуникациите е използван MPI. Числените експерименти са проведени на паралелни архитектури от различен тип както по отношение на броя

на процесорите, така и спрямо достъпа до паметта. Имената на машините (както ще се цитират в изложението), техните характеристики и местоположение са:

Parmac – клъстер от 4 двупроцесорни Power Macintosh компютри, свързани чрез Bay Stack 350 суич. Всеки възел има 512 MB RAM и два процесора Power PC G4/450MHz (ИПОИ – БАН, София).

Thea – клъстер от 8 компютъра, всеки със 768 MB RAM и един процесор AMD Athlon с честота 1.4GHz (IG AS CR, Ostrava, Czech Republic).

Beowulf – клъстер от 16 работни станции Digital (Compaq) Personal Workstations 500au свързани със 100Mbps switched Ethernet. Всеки възел е с един процесор EV56 на 500MHz и 128 MB RAM (TAMU, College Station, Texas, USA).

Grendel – Silicon Graphics Origin 2000 с 8 процесора R10000 на 250 MHz, 4 MB L2 кеш и 4GB RAM (TAMU, College Station, Texas, USA).

Blue – IBM SP (ASCI Blue) 5 с 336 възела и обща памет 512GB SDRAM. Всеки възел има 1536MB SDRAM и по 4 Power PC 604e процесора на 332MHz (LLNL, California, USA).

Grendel е с обща памет, Beowulf и Thea са с разпределена. При Parmac и Blue за комуникации между процесорите в един възел се използва обща памет, а когато са в различни – разпределена. При Parmac (виж E. I. Atanassov [10]) са възможни два начина на разпределение на процесите върху възлите при изпълнение на MPI програма. Те се осигуряват от опцията N на „mpirun“. При първия начин се натоварват максимално работещите възли. Всеки нов процес при втория начин се стартира върху неработещ възел докато има такива, а след това се натоварват последователно по същия начин. Поради големия общ брой процесори, втората възможност е забранена при Blue. Например, ако се използва „mpirun N -np 4 exec.out“, програмата exec.out ще се изпълни на 4 процесора намиращи се върху 4 различни възела – по 1 процесор на възел. В изложението този случай се означава с Parmac(N). Ако се използва „mpirun -np 4 exec.out“, т.е. „mpirun“ без опцията N (цитирано като Parmac), всички работещи процесори ще са върху два от възлите.

Глава 2

Преки методи за елиптични задачи с разделящи се променливи

Една от задачите на настоящата дисертация е да се реализират паралелно т. нар. бърз алгоритъм за разделяне на променливите и обобщен марш алгоритъм. В тази глава са разгледани подробно изходните последователни версии. По-точно, в първата част са описани стандартен и бърз алгоритъм за разделяне на променливите и е сравнена теоретично тяхната изчислителна сложност. Втората част е посветена на стандартен и обобщен марш алгоритъм, като отново е изследвана теоретично изчислителната им сложност. Въведена е и модификация на устойчивата версия, която има известни предимства по отношение на паралелните свойства. В третата част са изследвани експериментално устойчивостта и изчислителната ефективност на алгоритмите от първите два раздела. В последната част е представено едно приложение на преките методи за конструиране на преобусловители. Предложен е итерационен метод от тип спрегнат градиент за решаване на система линейни алгебрични уравнения получена след дискретизация на елиптични ЧДУ чрез неконформни четириъгълни крайни елементи върху завъртяна мрежа. Преобуславящата матрица е с разделящи се променливи. Конструирана е върху стандартна мрежа и системата с нея е решена чрез бързия алгоритъм за разделяне на променливите.

Резултатите, представени в тази глава са публикувани в [17, 18, 23].

2.1 Алгоритми за разделяне на променливите

В тази секция са описани два метода за числено решаване на двумерни елиптични задачи с разделящи се променливи върху равномерни $n \times m$ мрежи и е оценена

теоретично тяхната изчислителна сложност. По-точно това са стандартен дискретен алгоритъм за разделяне на променливите (SV) и бърз алгоритъм за разделяне на променливите (FASV), предложен от P.S. Vassilevski [93] и изследван по-късно от T. Rossi [79]. Подобрената изчислителна сложност на FASV се дължи на използваните блочна четно-нечетна елиминация и техника за непълно решаване на задачи с разредени десни части (SRHS). Последната е предложена независимо от A. Banegas [13], W. Proskurowski [76] и Y. Kuznetsov, A.M. Matsokin [67], като по-подробни теоретични изследвания на такива задачи са направени от Y. Kuznetsov [66]. SRHS е в основата не само на FASV, но и на марш алгоритмите описани в следващия параграф. Изложението в секцията следва съответната част от S. Petrova [75].

2.1.1 Дискретен метод за разделяне на променливите

За решаването на диференчната задача (1.1.5) чрез стандартния дискретен алгоритъм за разделяне на променливите (SV) се използва представянето ѝ (1.1.7).

Нека $\{\mathbf{q}_k, \lambda_k\}_{k=1}^m$ са ортонормираните собствени вектори на матрицата B и съответните им собствени стойности, т.е. $B\,\mathbf{q}_k = \lambda_k\,\mathbf{q}_k$, $k = 1, \ldots, m$. Тъй като B е симетрична и положително определена, то $\lambda_k > 0$ и $\mathbf{q}_k^T\,\mathbf{q}_r = \delta_{k,r}$, $k, r = 1, \ldots, m$, където $\delta_{k,r}$ е символа на Кронекер (Kronecker).

Векторите $\mathbf{x}_i'$ и $\mathbf{f}_i'$, вж. (1.1.7), $(i = 1, \ldots, n)$ се разлагат по базиса на ортонормалната система $\{\mathbf{q}_k\}_{k=1}^m$

$$\mathbf{x}_i' = \sum_{k=1}^{m} \eta_{i,k}\,\mathbf{q}_k\,, \quad \mathbf{f}_i' = \sum_{k=1}^{m} \beta_{i,k}\,\mathbf{q}_k\,, \quad i = 1, \ldots, n\,, \tag{2.1.1}$$

като коефициентите на Фурие (Fourier) $\beta_{i,k}$ за $\mathbf{f}_i'$ се определят чрез

$$\beta_{i,k} = \mathbf{f}_i'^T\,\mathbf{q}_k,\ i = 1, \ldots, n,\ k = 1, \ldots, m. \tag{2.1.2}$$

Решаването на системата (1.1.7) се свежда до намиране на коефициентите на Фурие $\eta_{i,k}$ за неизвестните вектори $\mathbf{x}_i'$.

Равенствата (2.1.1) се заместват в (1.1.7) и след опростяване се получава:

$$\left|\begin{array}{lcl}
\displaystyle\sum_{k=1}^{m} \left((\lambda_k + t_{1,1})\,\eta_{1,k} + t_{1,2}\,\eta_{2,k}\right)\,\mathbf{q}_k & = & \displaystyle\sum_{k=1}^{m} \beta_{1,k}\,\mathbf{q}_k \\
\ldots & & \\
\displaystyle\sum_{k=1}^{m} \left(t_{i,i-1}\,\eta_{i-1,k} + (\lambda_k + t_{i,i})\,\eta_{i,k} + t_{i,i+1}\,\eta_{i+1,k}\right)\,\mathbf{q}_k & = & \displaystyle\sum_{k=1}^{m} \beta_{i,k}\,\mathbf{q}_k \\
\ldots & & \\
\displaystyle\sum_{k=1}^{m} \left(t_{n,n-1}\,\eta_{n-1,k} + (\lambda_k + t_{n,n})\,\eta_{n,k}\right)\,\mathbf{q}_k & = & \displaystyle\sum_{k=1}^{m} \beta_{n,k}\,\mathbf{q}_k
\end{array}\right. . \tag{2.1.3}$$

Тъй като системата $\{\mathbf{q}_k\}_{k=1}^m$ е линейно независима и образува базис, то коефициентите пред съответните вектори от двете страни на уравненията в (2.1.3) трябва да са равни. Като се използват (за $k = 1, \ldots, m$) вектор-стълбовете $\eta_k = (\eta_{1,k}\,, \eta_{2,k}\,, \ldots\,, \eta_{n,k})^T$ и $\beta_k = (\beta_{1,k}\,, \beta_{2,k}\,, \ldots\,, \beta_{n,k})^T$, се получава система от независими едно от друго блочни уравнения с тридиагонални матрици:

$$(\lambda_k\, I_n + T)\, \eta_k = \beta_k,\ k = 1, \ldots, m\,.$$

Всяко от тях може да се реши по отношение на η_k чрез $LD^{-1}U$ факторизация, която в случая на тридиагонална матрица е добре известният метод на прогонката (вж. напр. [4, 11]).

Стъпките на дискретния алгоритъм за разделяне на променливите, заедно с необходимите аритметични операции (ар. оп.) за изпълнението им са:

Алгоритъм SV

Стъпка 0: *Намиране* на двойките $\{\mathbf{q}_k,\, \lambda_k\}_{k=1}^m$ за тридиагоналната матрица B (*изисква* $\mathcal{O}\,(m^3)$ *ар. оп.* – вж. Б. Н. Парлет [2]);

Стъпка 1: *Пресмятане* на коефициентите на Фурие $\beta_{i,k}$ за векторите $\mathbf{f}'_i$, използвайки (2.1.2) (*изисква* $2m^2n$ *ар. оп.*);

Стъпка 2: *Решаване* на m независими системи с тридиагонални матрици от ред n от вида $(\lambda_k\, I_n + T)\, \eta_k = \beta_k,\ k = 1, \ldots, m$, с цел да се получат коефициентите на Фурие $\eta_{i,k}$ (*изисква* $m(5n-4)$ *ар. оп. за решаване на* m *тридиагонални системи, като за точна* $LD^{-1}U$ *факторизация на матриците* $\lambda_k\, I_n + T,\ k = 1, \ldots, m$ *са необходими допълнително* $m(4n-3)$ *ар. оп.*);

Стъпка 3: *Възстановяване* на решението на задача (1.1.7) (или (1.1.6)) въз основа на коефициентите на Фурие $\{\eta_{i,k}\}_{i=1\,k=1}^{n\quad m}$ на векторите $\mathbf{x}'_i$. Възможно е *вертикално* или *хоризонтално* възстановяване съответно чрез $\mathbf{x}'_i = \sum_{k=1}^{m} \eta_{i,k}\, \mathbf{q}_k, i = 1, \ldots, n$ или $\mathbf{x}_j = \sum_{k=1}^{m} q_{j,k}\eta_k, j = 1, \ldots, m$ (*изисква* $2m^2n$ *ар. оп.*)

Край {SV}.

Общият брой операции, необходими за изпълнението на Алгоритъм SV се дава в следната известна теорема.

Теорема 2.1.1 *Алгоритъмът за разделяне на променливите за решаване на блочнотридиагонална система с размерност* $mn \times mn$ *изисква* $\mathcal{N}_{SV} \approx 4m^2n + m(5n-4)$ *ар. оп. в частта за решаване,* $m(4n-3)$ *ар. оп. за факторизация на тридиагонални матрици и* $\mathcal{O}\,(m^3)$ *ар. оп. за намиране на необходимите собствени вектори и собствени стойности.*

В случая $m = \mathcal{O}(n)$, който се среща често на практика, за изпълнение на алгоритъма са необходими $\mathcal{O}(n^3)$ ар. оп., т.е. $\mathcal{O}\left(N^{3/2}\right)$, където $N = mn$.

2.1.2 Непълно решаване на задачи с разредена дясна част

Предполага се, че дясната част $\mathbf{f}$ на задача (1.1.6) има само d ($d \ll m$) ненулеви блочни компоненти $\mathbf{f}_j$ и те са зададени с индексите $j_1, j_2, \ldots, j_d$, т.е. $\mathbf{f}_j = 0$ за $j \neq j_1, j_2, \ldots, j_d$. Тогава всеки блок $\mathbf{f}'_i, i = 1, \ldots, n$ от пренаредения вектор дясна част има по d ненулеви скаларни компоненти $f_{i,j_s}, s = 1, \ldots, d$. Допуска се също, че се търсят r ($r \ll m$) блочни компоненти $\mathbf{x}_j$ на решението $\mathbf{x}$ и нека те са $\mathbf{x}_{j'_1}, \mathbf{x}_{j'_2}, \ldots, \mathbf{x}_{j'_r}$. Следователно за $i = 1, \ldots, n$ трябва да се пресметнат само $j'_1, j'_2, \ldots, j'_r$-тите компоненти на всички $\mathbf{x}'_i$.

За да се намерят необходимите компоненти на решението, се прилага Алгоритъм SV с леки модификации:

Алгоритъм SRHS

Стъпка 0: *Намиране* на всички собствени стойности $\{\lambda_k\}_{k=1}^m$ и на необходимите $\widetilde{d} \leq d + r$ компоненти $\{\mathbf{q}_{k,j}\}$, $j \in \{j_1, \ldots, j_d\} \cup \{j'_1, \ldots, j'_r\}$ от всички собствени вектори $\{\mathbf{q}_k\}_{k=1}^m$ на тридиагоналната матрица B (*изисква* $\mathcal{O}\left(\widetilde{d}\, m^2\right)$ *ар. оп. за намиране на* $\widetilde{d}$ *компоненти на всички собствени вектори и* $9m^2$ *ар. оп. за намиране на всички собствени стойности, както е показано от Б.Н. Парлет [2]*);

Стъпка 1: *Изчисляване* на коефициентите на Фурие $\beta_{i,k}$ на $\mathbf{f}'_i$ от уравненията $\beta_{i,k} = \mathbf{q}_k^T \mathbf{f}'_i = \sum_{s=1}^{d} q_{j_s,k} f_{i,j_s}$, $i = 1, \ldots, n$, $k = 1, \ldots, m$ (*изисква* $2dmn$ *ар. оп., тъй като само* d *компоненти на* $\mathbf{f}'_i$ *са ненулеви*);

Стъпка 2: *Решаване* на m тридиагонални системи линейни уравнения от вида $(\lambda_k I_n + T)\,\eta_k = \beta_k$, $k = 1, \ldots, m$ (*изисква* $m(5n-4)$ *ар. оп. за решаване на* m *тридиагонални системи и допълнително* $m(4n-3)$ *ар. оп. за точна* $LD^{-1}U$ *факторизация на матриците* $\lambda_k I_n + T$, $k = 1, \ldots, m$);

Стъпка 3: *Възстановяване* на r компоненти на решението по редове чрез формулите $\mathbf{x}_j = \sum_{k=1}^{m} q_{j,k}\eta_k$, $j = j'_1, \ldots, j'_r$ (*изисква* $2rmn$ *ар. оп.*)

Край {SRHS}.

Изчислителната сложност на Алгоритъм SRHS се обобщава в следната известна теорема.

Теорема 2.1.2 *Алгоритъмът за разделяне на променливите за решаване на задача (1.1.6) с разредена дясна част, напр. за която* $\mathbf{f}_j = 0$ *за* $j \neq j_1, j_2, \ldots, j_d$ *и когато се търсят само част от компонентите на решението, именно* $\mathbf{x}_j$ *за* $j = j'_1, j'_2, \ldots, j'_r$ *изисква* $\mathcal{N}_{SRHS} \approx m[2(r+d)n + (5n-4)]$ *ар. оп. за решаване,* $m(4n-3)$ *ар. оп. за факторизация на тридиагоналните матрици* $\lambda_k I_n + T$ *и* $\mathcal{O}\left(\widetilde{d}m^2\right) + 9m^2$ *ар. оп. за намиране на всички собствени стойности и на* $\widetilde{d}$ *компоненти на всички собствени вектори на матрицата* B.

2.1.3 Бърз алгоритъм за разделяне на променливите

Бързият алгоритъм за разделяне на променливите (FASV) се състои от прав и обратен ход. Отново се разглежда задача (1.1.6), като за удобство при изложението се предполага, че $m = 2^l - 1$.

На всяка стъпка, както на правия така и на обратния ход, елементите на матрицата A се групират по подходящ начин. Нечетните блокове $A^{(k,s)}$ по диагонала на блочното представяне за стъпка k $(1 \leq k \leq l)$ се състоят от $2^k - 1$ блока от ред n:

$$A^{(k,s)} = I_{2^k-1} \otimes T + B_k^{(s)} \otimes I_n,\ s = 1, 2, \ldots, 2^{l-k}.$$

В последния израз, $B_k^{(s)}$ са следните главни подматрици на B $(s_k = (s-1)2^k)$:

$$B_k^{(s)} = \begin{pmatrix} b_{s_k+1,s_k+1} & b_{s_k+1,s_k+2} & 0 & \ldots & 0 \\ b_{s_k+2,s_k+1} & b_{s_k+2,s_k+2} & b_{s_k+2,s_k+3} & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & b_{s_k+2^k-1,s_k+2^k-2} & b_{s_k+2^k-1,s_k+2^k-1} \end{pmatrix}.$$

Четните компоненти $A^{(k)}_{2s,2s} = T + b_{s2^k,s2^k} I_n$ имат само по един блок от ред n.

Така за всяка стъпка k новото блочно представяне на матрицата A има вида:

$$A = \begin{pmatrix} A^{(k,1)} & A^{(k)}_{1,2} & 0 & \ldots & 0 \\ A^{(k)}_{2,1} & T + b_{2^k,2^k} I_n & A^{(k)}_{2,3} & \ldots & 0 \\ 0 & A^{(k)}_{3,2} & A^{(k,2)} & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & A^{(k)}_{2^{l-k+1}-1,2^{l-k+1}-2} & A^{(k,2^{l-k})} \end{pmatrix}. \tag{2.1.4}$$

Целта е на всяка стъпка на правия и обратния ход да се конструират системи, които могат да се решат чрез Алгоритъм SRHS.

Правият ход на FASV се състои от няколко стъпки. На първата стъпка се полага $\mathbf{f}^{(1)} = \mathbf{f}$. За да се осигури разредена блочна структура на дясната част, $\mathbf{f}^{(k+1)}$

(за $k = 1, \ldots, l-1$) се определя като вектор от остатъци:

$$\mathbf{f}^{(k+1)} = \mathbf{f}^{(k)} - A\mathbf{x}^{(k)}, (\mathbf{f}^{(1)} = \mathbf{f}). \tag{2.1.5}$$

Векторът $\mathbf{x}^{(k)}$, за всяко $k = 1, \ldots, l-1$, се конструира по следния начин: Решават се задачите

$$A^{(k,s)}\,\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)},\ s = 1, 2, \ldots, 2^{l-k}, \tag{2.1.6}$$

където

$$\mathbf{x}^{(k,s)} = \begin{pmatrix} \mathbf{x}^{(k)}_{s_k+1} \\ \mathbf{x}^{(k)}_{s_k+2} \\ \vdots \\ \mathbf{x}^{(k)}_{s_k+2^k-1} \end{pmatrix} \text{ и } \mathbf{f}^{(k,s)} = \begin{pmatrix} \mathbf{0} & \} & 2^{k-1}-1 & \text{блока} \\ \mathbf{f}^{(k)}_{s_k+2^{k-1}} & \} & 1 & \text{блок} \\ \mathbf{0} & \} & 2^{k-1}-1 & \text{блока} \end{pmatrix}.$$

Полага се $\mathbf{x}^{(k)}_{s.2^k} = 0$ за $s = 1, 2, \ldots, 2^{l-k}-1$. Векторите

$$\mathbf{x}^{(k)} = \begin{pmatrix} \mathbf{x}^{(k,1)} & \} & 2^k-1 & \text{блока} \\ \mathbf{0} & \} & 1 & \text{блок} \\ \mathbf{x}^{(k,2)} & \} & 2^k-1 & \text{блока} \\ \mathbf{0} & \} & 1 & \text{блок} \\ \vdots & & \vdots & \\ \mathbf{x}^{(k,2^{l-k})} & \} & 2^k-1 & \text{блока} \end{pmatrix}, \ \mathbf{f}^{(k)} = \begin{pmatrix} \mathbf{f}^{(k,1)} & \} & 2^k-1 & \text{блока} \\ \mathbf{f}^{(k)}_{2^k} & \} & 1 & \text{блок} \\ \mathbf{f}^{(k,2)} & \} & 2^k-1 & \text{блока} \\ \mathbf{f}^{(k)}_{2.2^k} & \} & 1 & \text{блок} \\ \vdots & & \vdots & \\ \mathbf{f}^{(k,2^{l-k})} & \} & 2^k-1 & \text{блока} \end{pmatrix} \tag{2.1.7}$$

се заместват в (2.1.5) и след преобразуване, за вектора $\mathbf{f}^{(k+1)}$ се получава

$$\mathbf{f}^{(k+1)} = \begin{pmatrix} \mathbf{0} & \} & 2^k-1 & \text{блока} \\ \mathbf{f}^{(k+1)}_{1.2^k} & \} & 1 & \text{блок} \\ \mathbf{0} & \} & 2^k-1 & \text{блока} \\ \mathbf{f}^{(k+1)}_{2.2^k} & \} & 1 & \text{блок} \\ \vdots & & \vdots & \\ \mathbf{f}^{(k+1)}_{(2^{l-k}-1).2^k} & \} & 1 & \text{блок} \\ \mathbf{0} & \} & 2^k-1 & \text{блока} \end{pmatrix}.$$

По-точно, $\mathbf{f}^{(k+1)}$ може да има ненулеви блочни компоненти $\mathbf{f}^{(k+1)}_i$ само за $i = s.2^k$, $s = 1, 2, \ldots, 2^{l-k}-1$. Тъй като $A^{(k)}_{2s,2s-1} = (0, \ldots, 0, b_{s_k+2^k, s_k+2^k-1} I_n)$ и $A^{(k)}_{2s,2s+1} = (b_{s_k+2^k, s_k+2^k+1} I_n, 0, \ldots, 0)$, те се определят чрез:

$$\begin{aligned} \mathbf{f}^{(k+1)}_{s.2^k} &= \mathbf{f}^{(k)}_{s.2^k} - A^{(k)}_{2s,2s-1}\mathbf{x}^{(k,s)} - A^{(k)}_{2s,2s+1}\mathbf{x}^{(k,s+1)} \\ &= \mathbf{f}^{(k)}_{s.2^k} - b_{s.2^k, s.2^k-1}\mathbf{x}^{(k,s)}_{2^k-1} - b_{s.2^k, s.2^k+1}\mathbf{x}^{(k,s+1)}_1 \\ &= \mathbf{f}^{(k)}_{s.2^k} - b_{s.2^k, s.2^k-1}\mathbf{x}^{(k)}_{s.2^k-1} - b_{s.2^k, s.2^k+1}\mathbf{x}^{(k)}_{s.2^k+1}. \end{aligned} \tag{2.1.8}$$

Въвежда се означението

$$\mathbf{f}^{(k+1,s')} = \begin{pmatrix} \mathbf{0} \\ \mathbf{f}^{(k+1)}_{s.2^k} \\ \mathbf{0} \end{pmatrix} \begin{matrix} \} & 2^k-1 & \text{блока} \\ \} & 1 & \text{блок} \\ \} & 2^k-1 & \text{блока} \end{matrix} ,$$

където $s = 2s' - 1,\ s' = 1, 2, \ldots, 2^{l-k-1}$ и $\mathbf{f}^{(k+1)}$ получава вида на $\mathbf{f}^{(k)}$ в (2.1.7). По този начин броят на ненулевите компоненти на $\mathbf{f}^{(k)}$ на всяка стъпка намалява наполовина.

Пример: При $l = 4$, $m = 2^4 - 1 = 15$, векторите $\mathbf{f}^{(k)}$, $k = 1, \ldots 4$ ще имат следните ненулеви блокове:

$\mathbf{f}^{(1)}$:	$\mathbf{f}^{(1)}_1$	$\mathbf{f}^{(1)}_2$	$\mathbf{f}^{(1)}_3$	$\mathbf{f}^{(1)}_4$	$\mathbf{f}^{(1)}_5$	$\mathbf{f}^{(1)}_6$	$\mathbf{f}^{(1)}_7$	$\mathbf{f}^{(1)}_8$	$\mathbf{f}^{(1)}_9$	$\mathbf{f}^{(1)}_{10}$	$\mathbf{f}^{(1)}_{11}$	$\mathbf{f}^{(1)}_{12}$	$\mathbf{f}^{(1)}_{13}$	$\mathbf{f}^{(1)}_{14}$	$\mathbf{f}^{(1)}_{15}$
$\mathbf{f}^{(2)}$:		$\mathbf{f}^{(2)}_2$		$\mathbf{f}^{(2)}_4$		$\mathbf{f}^{(2)}_6$		$\mathbf{f}^{(2)}_8$		$\mathbf{f}^{(2)}_{10}$		$\mathbf{f}^{(2)}_{12}$		$\mathbf{f}^{(2)}_{14}$	
$\mathbf{f}^{(3)}$:				$\mathbf{f}^{(3)}_4$				$\mathbf{f}^{(3)}_8$				$\mathbf{f}^{(3)}_{12}$			
$\mathbf{f}^{(4)}$:								$\mathbf{f}^{(4)}_8$							

За пресмятане на ненулевите компоненти на $\mathbf{f}^{(k+1)}$ са необходими по два блока на всяко $\mathbf{x}^{(k,s)}$, именно $\mathbf{x}^{(k,s)}_1$ и $\mathbf{x}^{(k,s)}_{2^k-1}$ (вж. (2.1.8)). При възстановяване на решението на обратния ход ще бъде необходима още компонентата $\mathbf{x}^{(k,s)}_{2^{k-1}}$.

Така конструираните системи $A^{(k,s)}\,\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)},\ s = 1, 2, \ldots, 2^{l-k}$ имат разредена дясна част с една ненулева блочна компонента $(\mathbf{f}^{(k,s)}_{2^k-1})$ и се търсят три блочни компоненти на решението. Те могат да се решат непълно като се използва Алгоритъм SRHS с $d = 1$, $r = 3$ и $m = 2^k - 1$.

Забележка: Матриците $A^{(k,s)}$ позволяват разделяне на променливите, защото са от същия вид като изходната матрица A, но B е заменена с нейна главна подматрица $B^{(s)}_k$ (вж. (2.1.4)).

При конструирането на **обратния ход на FASV** са използвани следните факти:

1) Решението на системата $A\mathbf{x} = \mathbf{f}$ може да се разложи като сума на векторите $\mathbf{x}^{(k)}$, $k = 1, \ldots, l$:

$$\mathbf{x} = \mathbf{x}^{(1)} + \mathbf{x}^{(2)} + \cdots + \mathbf{x}^{(l)}. \tag{2.1.9}$$

Проверката е следната: $\mathbf{f}^{(k+1)}$ в тъждеството $\mathbf{f}^{(1)} - \mathbf{f}^{(l+1)} = \sum_{k=1}^{l}(\mathbf{f}^{(k)} - \mathbf{f}^{(k+1)})$ се замества с равното му от (2.1.5) и след преобразуване се получава, че

$$\mathbf{f}^{(1)} - \mathbf{f}^{(l+1)} = \sum_{k=1}^{l} A\mathbf{x}^{(k)} = A\left(\sum_{k=1}^{l}\mathbf{x}^{(k)}\right).$$

Тъй като $\mathbf{f}^{(1)} = \mathbf{f}$ и от конструкцията $\mathbf{f}^{(l+1)} \equiv \mathbf{f}^{(l)} - A\mathbf{x}^{(l)} \equiv \mathbf{f}^{(l,1)} - A^{(l,1)}\mathbf{x}^{(l,1)} = 0$, то (2.1.9) е в сила.

2) Според конструкцията $\mathbf{x}^{(k)}_{s.2^k} = 0$ за $s = 1, 2, \ldots, 2^{l-k} - 1$. Това равенство означава, че при нарастване на k, броят на нулевите компоненти на $\mathbf{x}^{(k)}$ намалява наполовина.

Пример: При $l = 4$, $m = 2^4 - 1 = 15$, векторите $\mathbf{x}^{(k)}$ имат нулеви компоненти на следните места:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{x}^{(1)}$:		0		0		0		0		0		0		0	
$\mathbf{x}^{(2)}$:				0				0				0			
$\mathbf{x}^{(3)}$:								0							
$\mathbf{x}^{(4)}$:															

С други думи $\mathbf{x}^{(k)}_{s.2^{k_0}} = 0$ за $k \leq k_0$. От тук и (2.1.9) следва, че за точното решение $\mathbf{x}$ е в сила

$$\mathbf{x}_{s.2^{k_0}} = \left(\sum_{k=1}^{l} \mathbf{x}^{(k)} \right)_{s.2^{k_0}} = \sum_{k>k_0} \mathbf{x}^{(k)}_{s.2^{k_0}}. \tag{2.1.10}$$

При обратния ход на FASV целта е за $k = l, l-1, \ldots, 1$ и $s = 1, 2, \ldots, 2^{l-k}$ да се определят $\mathbf{x}_{(2s-1)2^{k-1}}$. Така на всяка стъпка в обратния ход броят на търсените компоненти на $\mathbf{x}$ се увеличава, докато се възстанови целия вектор $\mathbf{x}$.

Пример: При $l = 4$, $m = 2^4 - 1 = 15$, на всяка стъпка k се търсят $\mathbf{x}_{(2s-1)2^{k-1}}$ за $s = 1, 2, \ldots, 2^{4-k}$. Решението се възстановява на части по следния начин:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$k = 4$:								$\mathbf{x}_8$							
$k = 3$:				$\mathbf{x}_4$				$*$				$\mathbf{x}_{12}$			
$k = 2$:		$\mathbf{x}_2$		$*$		$\mathbf{x}_6$		$*$		$\mathbf{x}_{10}$		$*$		$\mathbf{x}_{14}$	
$k = 1$:	$\mathbf{x}_1$	$*$	$\mathbf{x}_3$	$*$	$\mathbf{x}_5$	$*$	$\mathbf{x}_7$	$*$	$\mathbf{x}_9$	$*$	$\mathbf{x}_{11}$	$*$	$\mathbf{x}_{13}$	$*$	$\mathbf{x}_{15}$

Символът $*$ означава, че съответната компонента вече е намерена.

На първата стъпка на обратния ход, т.е. когато $k = l$, ($2^{(l-k)} = 1$, $s = 1$), трябва да се намери една блочна компонента на решението, а именно $\mathbf{x}_{(2s-1)2^{k-1}} \equiv \mathbf{x}_{2^{l-1}}$. От друга страна, от равенството (2.1.10) за $k_0 = l - 1$ и $s = 1$ следва, че $\mathbf{x}_{2^{l-1}} = \mathbf{x}^{(l)}_{2^{l-1}}$. Това означава, че средната блочна компонента на решението $\mathbf{x}$ на системата $A\mathbf{x} = \mathbf{f}$ се намира чрез непълно решаване (с $r = 1$) на системата

$$A\mathbf{x}^{(l)} = \mathbf{f}^{(l)}, \tag{2.1.11}$$

където $A \equiv A^{(l,1)}$, $\mathbf{x}^{(l)} = \mathbf{x}^{(l,1)}$ и $\mathbf{f}^{(l)} = \mathbf{f}^{(l,1)}$. Дясната част $\mathbf{f}^{(l)}$ е намерена на последната стъпка на правия ход и има разредена структура с една ненулева блочна компонента, т.е. $d = 1$. (Това следва от уравнение (2.1.8).)

За останалите стъпки $k = l-1, \ldots, 1$ се конструират системи, чието непълно решаване води до намиране на останалите компоненти на $\mathbf{x}$. От (2.1.5) следва

$$\begin{aligned} A\sum_{i=k}^{l}\mathbf{x}^{(i)} &= \sum_{i=k}^{l}A\mathbf{x}^{(i)} = \sum_{i=k}^{l}\left(\mathbf{f}^{(i)}-\mathbf{f}^{(i+1)}\right) \\ &= \mathbf{f}^{(k)}-\mathbf{f}^{(k+1)}+\mathbf{f}^{(k+1)}-\mathbf{f}^{(k+2)}+\cdots+\mathbf{f}^{(l-1)}-\mathbf{f}^{(l)}+\mathbf{f}^{(l)}-\mathbf{f}^{(l+1)} \\ &= \mathbf{f}^{(k)}-\mathbf{f}^{(l+1)} = \mathbf{f}^{(k)}, \end{aligned}$$

т.е.

$$A\sum_{i=k}^{l}\mathbf{x}^{(i)} = \mathbf{f}^{(k)}. \tag{2.1.12}$$

Ако за A, $z^{(k)} = \sum_{i=k}^{l}\mathbf{x}^{(i)}$ и $\mathbf{f}^{(k)}$ се използват блочните представяния (2.1.4),

$$\mathbf{z}^{(k)^T} = (\mathbf{z}^{(k,1)^T}, \mathbf{z}_{2^k}^{(k)^T}, \ldots, \mathbf{z}^{(k,2^{l-k})^T}) \text{ и } \mathbf{f}^{(k)^T} = (\mathbf{f}^{(k,1)^T}, \mathbf{f}_{2^k}^{(k)^T}, \ldots, \mathbf{f}^{(k,2^{l-k})^T}),$$

то матричното уравнение (2.1.12) се записва във вида

$$\left|\begin{array}{lcl} A^{(k,1)}\mathbf{z}^{(k,1)} + A_{1,2}^{(k)}\mathbf{z}_{2^k}^{(k)} & = & \mathbf{f}^{(k,1)} \\ A_{2,1}^{(k)}\mathbf{z}^{(k,1)} + (T+b_{2^k,2^k}I_n)\mathbf{z}_{2^k}^{(k)} + A_{2,3}^{(k)}\mathbf{z}^{(k,2)} & = & \mathbf{f}_{2^k}^{(k)} \\ \ldots & & \\ A_{2s-1,2s-2}^{(k)}\mathbf{z}_{(s-1)2^k}^{(k)} + A^{(k,s)}\mathbf{z}^{(k,s)} + A_{2s-1,2s}^{(k)}\mathbf{z}_{s.2^k}^{(k)} & = & \mathbf{f}^{(k,s)} \\ A_{2s,2s-1}^{(k)}\mathbf{z}^{(k,s)} + (T+b_{s.2^k,s.2^k}I_n)\mathbf{z}_{s.2^k}^{(k)} + A_{2s,2s+1}^{(k)}\mathbf{z}^{(k,s+1)} & = & \mathbf{f}_{s.2^k}^{(k)} \\ \ldots & & \\ A_{2^{l-k+1}-1,2^{l-k+1}-2}^{(k)}\mathbf{z}_{(2^{l-k}-1)2^k}^{(k)} + A^{(k,2^{l-k})}\mathbf{z}^{(k,2^{l-k})} & = & \mathbf{f}^{(k,2^{l-k})} \end{array}\right. .$$

Като се вземат подсистемите с матрици $A^{(k,s)}$ или, което е едно и също, за някое дадено $s \geq 1$ и $s_k = (s-1)2^k$, (s_k+1)-то, (s_k+2)-то, ... , (s_k+2^k-1)-то уравнение от тъждеството (2.1.12) се получава

$$\mathbf{f}^{(k,s)} = A^{(k,s)}\mathbf{z}^{(k,s)} + A_{2s-1,2s-2}^{(k)}\mathbf{z}_{(s-1)2^k}^{(k)} + A_{2s-1,2s}^{(k)}\mathbf{z}_{s.2^k}^{(k)}. \tag{2.1.13}$$

От (2.1.10) за $s-1$ и s следва $\sum_{k'>k}\mathbf{x}_{s_k}^{(k')} = \mathbf{x}_{s_k}$, $\sum_{k'>k}\mathbf{x}_{s.2^k}^{(k')} = \mathbf{x}_{s.2^k}$ и тъй като $\mathbf{x}_{s_k}^{(k)} = 0$ и $\mathbf{x}_{s.2^k}^{(k)} = 0$, то $\mathbf{z}_{(s-1)2^k}^{(k)} = \sum_{k'\geq k}\mathbf{x}_{s_k}^{(k')} = \mathbf{x}_{s_k}$ и $\mathbf{z}_{s.2^k}^{(k)} = \sum_{k'\geq k}\mathbf{x}_{s.2^k}^{(k')} = \mathbf{x}_{s.2^k}$. Полага се $\mathbf{y}_r^{(k)} = \sum_{k'\geq k}\mathbf{x}_{s_k+r}^{(k')}$ за $r = 1, \ldots, 2^k-1$ (т.е. $\mathbf{y}^{(k)} = \mathbf{z}^{(k,s)}$) и (2.1.13) се записва във вида

$$A^{(k,s)}\mathbf{y}^{(k)} = \mathbf{f}^{(k,s)} - A_{2s-1,2s-2}^{(k)}\mathbf{x}_{s_k} - A_{2s-1,2s}^{(k)}\mathbf{x}_{s.2^k}. \tag{2.1.14}$$

Блоковете $A^{(k)}_{2s-1,2s-2}$ и $A^{(k)}_{2s-1,2s}$ са

$$\begin{aligned} A^{(k)}_{2s-1,2s-2} = A^{(k)}_{2(s-1)+1,2(s-1)} &= (b_{(s-1)2^k+1,(s-1)2^k} I_n\,,\, \mathbf{0}\,,\, \ldots\,,\, \mathbf{0})^T\,, \quad \text{за } k>1 \\ A^{(k)}_{2s-1,2s} &= (\mathbf{0}\,,\, \ldots\,,\, \mathbf{0}\,,\, b_{s.2^k-1,s.2^k} I_n)^T\,, \\ A^{(k)}_{2s-1,2s-2} &= b_{(s-1)2^k+1,(s-1)2^k} I_n\,, \qquad \text{за } k=1 \\ A^{(k)}_{2s-1,2s} &= b_{s.2^k-1,s.2^k} I_n\,. \end{aligned}$$

Следователно (2.1.14) може да се запише във вида

$$A^{(k,s)}\mathbf{y}^{(k)} = \mathbf{f}^{(k,s)} + \widehat{\mathbf{f}}^{(k,s)}, \tag{2.1.15}$$

където

$$\widehat{\mathbf{f}}^{(k,s)} = \begin{cases} \begin{pmatrix} -b_{s_k+1,s_k}\mathbf{x}_{s_k} \\ 0 \\ \vdots \\ 0 \\ -b_{s.2^k-1,s.2^k}\mathbf{x}_{s.2^k} \end{pmatrix}, & k \neq 1 \\ -b_{s_k+1,s_k}\mathbf{x}_{s_k} - b_{s.2^k-1,s.2^k}\mathbf{x}_{s.2^k}\,, & k=1 \end{cases}.$$

На всяка стъпка на обратния ход се решава непълно система от вида (2.1.15), като се търси само

$$\mathbf{y}^{(k)}_{2^{k-1}} = \sum_{k'\geq k} \mathbf{x}^{(k')}_{(s-1)2^k+2^{k-1}} = \sum_{k'\geq k} \mathbf{x}^{(k')}_{(2s-2+1)2^{k-1}} = \sum_{k'\geq k} \mathbf{x}^{(k')}_{(2s-1)2^{k-1}} = \mathbf{x}_{(2s-1)2^{k-1}}\,.$$

От друга страна, $\mathbf{x}^{(k,s)}$ е решение на системата $A^{(k,s)}\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)}$ и $\mathbf{y}^{(k)}$ може да се разложи като $\mathbf{y}^{(k)} = \widehat{\mathbf{y}}^{(k)} + \mathbf{x}^{(k,s)}$, където $\widehat{\mathbf{y}}^{(k)}$ е решение на $A^{(k,s)}\widehat{\mathbf{y}}^{(k)} = \widehat{\mathbf{f}}^{(k,s)}$. Следователно $\mathbf{y}^{(k)}_{2^{k-1}} = \widehat{\mathbf{y}}^{(k)}_{2^{k-1}} + \mathbf{x}^{(k,s)}_{2^{k-1}}$, като $\mathbf{x}^{(k,s)}_{2^{k-1}}$ вече е намерен при правия ход.

Така конструираните системи $A^{(k,s)}\widehat{\mathbf{y}}^{(k)} = \widehat{\mathbf{f}}^{(k,s)}$ за всяка стъпка на обратния ход, се решават чрез Алгоритъм SRHS с $d=2$ и $r=1$. Компонентите $\mathbf{x}_{s_k}$ и $\mathbf{x}_{s.2^k}$, необходими за образуване на $\widehat{\mathbf{f}}^{(k,s)}$ са намерени на предишните стъпки. За $k=1$ получените системи имат една блочна компонента и се решават пълно.

Бързият алгоритъм за разделяне на променливите (FASV) може да бъде описан накратко по следния начин:

Алгоритъм FASV

Стъпка 1. Прав ход:

Полага се $\mathbf{f}^{(1)} = \mathbf{f}$

for $k = 1$ to $l-1$

for $s = 1$ to 2^{l-k} *се решават* системите

(2.1.16) $$A^{(k,s)}\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)}$$

непълно, като се намират само $\mathbf{x}_1^{(k,s)}$, $\mathbf{x}_{2^{k-1}}^{(k,s)}$, $\mathbf{x}_{2^k-1}^{(k,s)}$

end{цикъл по s}

for $s = 1$ to $2^{l-k} - 1$ *се изчисляват* векторите

(2.1.17) $$\mathbf{f}_{s.2^k}^{(k+1)} = \mathbf{f}_{s.2^k}^{(k)} - b_{s.2^k,s.2^k-1}\mathbf{x}_{2^k-1}^{(k,s)} - b_{s.2^k,s.2^k+1}\mathbf{x}_1^{(k,s+1)}$$

end {цикъл по s}

end {цикъл по k}

Стъпка 2. Обратен ход:

Системата $A\mathbf{x}^{(l)} = \mathbf{f}^{(l)}$ *се решава* непълно, само за $\mathbf{x}_{2^{l-1}}^{(l)} = \mathbf{x}_{2^{l-1}}$

for $k = l - 1$ downto 1

for $s = 1$ to 2^{l-k} *се пресмята* дясната част

$$\widehat{\mathbf{f}}^{(k,s)} = \begin{cases} \begin{pmatrix} -b_{s_k+1,s_k}\mathbf{x}_{s_k} \\ 0 \\ \vdots \\ -b_{s.2^k-1,s.2^k}\mathbf{x}_{s.2^k} \end{pmatrix}, & k \neq 1 \\ -b_{s_k+1,s_k}\mathbf{x}_{s_k} - b_{s.2^k-1,s.2^k}\mathbf{x}_{s.2^k}, & k = 1 \end{cases}$$

и *се решават* системите

(2.1.18) $$A^{(k,s)}\widehat{\mathbf{y}}^{(k)} = \widehat{\mathbf{f}}^{(k,s)}$$

непълно само за $\widehat{\mathbf{y}}_{2^{k-1}}^{(k)}$. След това се *полага*

(2.1.19) $$\mathbf{x}_{(2s-1)2^{k-1}} = \widehat{\mathbf{y}}_{2^{k-1}}^{(k)} + \mathbf{x}_{2^{k-1}}^{(k,s)}$$

end {цикъл по s}

end {цикъл по k}

Край {FASV}.

Броят на ар. оп., необходими за изпълнение на правия и обратния ход и на целия Алгоритъм FASV, се определя с помощта на **Теорема 2.1.2**. Известна е следната теорема.

Теорема 2.1.3 *Правият ход на* Алгоритъм FASV *изисква*

$$\sum_{k=1}^{l-1} 2^{l-k}(4\,n + (13\,n-4)\,n_k) \sim 13\,n\,m(l-1) - 9\,n\,m$$

ар. оп. в частта за решаване и

$$\sum_{k=1}^{l-1} 2^{l-k} n_k\,(4n-3) \sim 4\,n\,m\,(l-1)$$

ар. оп. за факторизация на матриците $T + \lambda_k I_n$, *където* $n_k = 2^k - 1$, $m = 2^l - 1$.

Изчислителната сложност на обратния ход на FASV е

$$\sum_{k=1}^{l-1} 2^{l-k}(2\,n + (11\,n-4)\,n_k) + 9\,n\,m - 4\,m \sim 11\,n\,m(l-1)\,.$$

Общо, прилагането на Алгоритъм FASV *за решаване на задачата* $A\mathbf{x} = \mathbf{f}$ *изисква* $\mathcal{N}_{FASV} \approx 24nm(l-1) - 9nm$ *ар. оп. в частта за решаване и* $4nm(l-1)$ *ар. оп. за факторизация на всички матрици* $T + \lambda_k I_n$ *в произведения от вида* $L_k D_k^{-1} U_k$, *където* L_k, U_k *са съответно долнотриъгълни и горнотриъгълни с единици по главния диагонал, а* D_k *е диагонална матрица.*

Доказателство. *За правия ход:* Множителят 2^{l-k} дава броя на решаваните системи от вида (2.1.16). Пресмятането на дясната страна $\mathbf{f}^{(k+1)}$ за всяка следваща стъпка посредством (2.1.17) изисква $4n$ ар. оп. Частта $(13n-4)\,n_k$ дава броя ар. оп. за непълно решаване на (2.1.16), намирайки само $\mathbf{x}_1^{(k,s)}, \mathbf{x}_{2^{k-1}}^{(k,s)}$ и $\mathbf{x}_{2^k-1}^{(k,s)}$ чрез Алгоритъм SRHS. Тя се получава, като се приложи **Теорема 2.1.2** при $r = 3$ (брой търсени компоненти на $\mathbf{x}^{(k,s)}$) и $d = 1$ (брой ненулеви компоненти на дясната част $\mathbf{f}^{(k,s)}$) и като се отчете, че се търсят собствените стойности и вектори на $B_k^{(s)}$ от ред $n_k = 2^k - 1$, вместо на B.

За обратния ход: Множителят 2^{l-k} е заради броя на задачите (2.1.18), за изчисляването на дясната страна на (2.1.18) са необходими $2n$ ар. оп. Непълното решаване на (2.1.18) изисква $(11\,n-4)n_k$ ар. оп., пресметнати с помощта на **Теорема 2.1.2** за $r = 1$ (брой търсени компоненти на решението) и $d = 2$ (брой ненулеви компоненти на дясната част).

За непълното решаване на задача $A\mathbf{x}^{(l)} = \mathbf{f}^{(l)}$ с $d = 1$ и $r = 1$ са необходими $9\,n\,m - 4\,m$ ар. оп. ∎

Забележка: За по-голяма точност, към предишната изчислителна сложност трябва да се добави и броят на операциите, необходими за пресмятане на всички собствени стойности и 1-та, 2^{k-1}-та, и (2^k-1)-та компонента на всички собствени вектори

на тридиагоналните матрици $B_k^{(s)}$ от ред n_k ($n_k = 2^k - 1$) за $s = 1, 2, \ldots, 2^{l-k}$ и $k = 1, \ldots, l$. По алгоритъма предложен в Б. Н. Парлет [2], тези данни се намират за $\sum_{k=1}^{l} 2^{l-k} \mathcal{O}\left(n_k^2\right) = \mathcal{O}\left(n^2\right)$ ар. оп., т.е. това не е доминираща част в изчислителната сложност на алгоритъма.

2.2 Условно устойчиви марш алгоритми

В тази секция са разгледани стандартен марш алгоритъм (SM) и обобщен марш алгоритъм (GMA) за решаване на системи уравнения от вида (1.1.6), получени след дискретизация на елиптични задачи с разделящи се променливи върху равномерна $n \times m$ мрежа. Тези два алгоритъма са предложени най-напред от R. Bank и D. Rose [14, 16], а по-късно са преформулирани от P.S. Vassilevski [94] с помощта на техниката за непълно решаване на задачи с разредена дясна част. В края на секцията е въведена и модификация на GMA (предложена от G. Bencheva [20]), чиито паралелни свойства са анализирани в следващата глава. Получени са оценки за максималния брой аритметични операции, необходими за изпълнението на SM и двата варианта на GMA. Изложението е базирано на резултатите в P.S. Vassilevski [94] и G. Bencheva [20].

2.2.1 Стандартен марш алгоритъм

В основата както на стандартния, така и на обобщения марш алгоритми е пренареждането и записването на матрицата на изходната задача (1.1.6) в блочна 2×2 форма със специфична структура. Получената система се решава чрез блочна Гаусова елиминация (вж. т. 1.2.1). При стандартния марш алгоритъм (SM) първото блочно уравнение се записва на последно място и пренареденатa система има вида:

$$\begin{pmatrix} U & G \\ C & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{x}_m \end{pmatrix} = \begin{pmatrix} \mathbf{f}' \\ \mathbf{f}_1 \end{pmatrix}, \tag{2.2.20}$$

където U е горнотриъгълната матрица

$$U = \begin{pmatrix} b_{2,1}I_n & T + b_{2,2}I_n & b_{2,3}I_n & \ldots & 0 \\ 0 & b_{3,2}I_n & T + b_{3,3}I_n & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & b_{m-1,m-2}I_n & T + b_{m-1,m-1}I_n \\ 0 & \ldots & \ldots & 0 & b_{m,m-1}I_n \end{pmatrix},$$

а матриците G и C и векторите $\mathbf{x}'$ и $\mathbf{f}'$ се определят чрез:

$$G = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ b_{m-1,m} I_n \\ T + b_{m,m} I_n \end{pmatrix}, \qquad \begin{aligned} C &= \left(T + b_{1,1} I_n \, , \, b_{1,2} I_n \, , \, 0 \, , \, \ldots \, , \, 0\right), \\ \mathbf{x}'^T &= \left(\mathbf{x}_1^T \, , \, \mathbf{x}_2^T \, , \, \ldots \, , \, \mathbf{x}_{m-1}^T\right), \\ \mathbf{f}'^T &= \left(\mathbf{f}_2^T \, , \, \mathbf{f}_3^T \, , \, \ldots \, , \, \mathbf{f}_m^T\right). \end{aligned}$$

За пренареденатa матрица е в сила следната блочна факторизация

$$\begin{pmatrix} U & G \\ C & 0 \end{pmatrix} = \begin{pmatrix} U & 0 \\ C & -CU^{-1}G \end{pmatrix} \begin{pmatrix} I & U^{-1}G \\ 0 & I \end{pmatrix}$$

и системата (2.2.20) добива вида:

$$\begin{pmatrix} U & 0 \\ C & -CU^{-1}G \end{pmatrix} \begin{pmatrix} I & U^{-1}G \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{x}_m \end{pmatrix} = \begin{pmatrix} \mathbf{f}' \\ \mathbf{f}_1 \end{pmatrix}. \tag{2.2.21}$$

За решаването на (2.2.21) се прилага блочната Гаусова елиминация (описана в т. 1.2.1) с $A_{1,1} \equiv U$, $A_{1,2} \equiv G$, $A_{2,1} \equiv C$, $A_{2,2} \equiv 0$, $S = -CU^{-1}G$ и съответните им компоненти $\mathbf{x}'$, $\mathbf{x}_m$, $\mathbf{f}'$, $\mathbf{f}_1$. Така се получава, че (2.2.20) е еквивалентна на системата:

$$\left| \begin{aligned} &U\mathbf{y}_1 = \mathbf{f}' \\ &-CU^{-1}G\mathbf{x}_m = \mathbf{f}_1 - C\mathbf{y}_1 = \mathbf{f}_1 - CU^{-1}\mathbf{f}' \\ &U\mathbf{x}' = -G\mathbf{x}_m + \mathbf{f}' \end{aligned} \right. . \tag{2.2.22}$$

Системите с горнотриъгълната матрица U се решават чрез рекурентната връзка на стандартния обратен ход, описана накратко по-долу.

Разглежда се системата

$$U \begin{pmatrix} \underline{\xi}_1 \\ \underline{\xi}_2 \\ \vdots \\ \underline{\xi}_{m-1} \end{pmatrix} = \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_{m-1} \end{pmatrix},$$

където $\underline{\xi}_i$, $\mathbf{g}_i \in \mathbb{R}^n$, $i = 1, \ldots, m-1$ или по компонентно

$$\begin{aligned} b_{2,1}\underline{\xi}_1 + (b_{2,2}I_n + T)\underline{\xi}_2 + b_{2,3}\underline{\xi}_3 &= \mathbf{g}_1 \\ \ddots \qquad\qquad &\vdots \\ b_{m-1,m-2}\underline{\xi}_{m-2} + (b_{m-1,m-1}I_n + T)\underline{\xi}_{m-1} &= \mathbf{g}_{m-2} \\ b_{m,m-1}\underline{\xi}_{m-1} &= \mathbf{g}_{m-1} \end{aligned} .$$

Решението $\underline{\xi}$ се получава по формулите:

$$
\begin{aligned}
&\underline{\xi}_{m-1} = \frac{1}{b_{m,m-1}}\mathbf{g}_{m-1};\\
&\text{for } i = m-1 \text{ down to } 2\\
&\qquad \underline{\xi}_{i-1} = \frac{1}{b_{i,i-1}}\left(\mathbf{g}_{i-1} - b_{i,i}\underline{\xi}_i - T\underline{\xi}_i - b_{i,i+1}\underline{\xi}_{i+1}\right)\\
&\text{end}\,.
\end{aligned}
\tag{2.2.23}
$$

Системата с допълнението на Шур $S = -CU^{-1}G$

$$
-CU^{-1}G\mathbf{x}_m = \widetilde{\mathbf{f}}_1 \equiv \mathbf{f}_1 - CU^{-1}\mathbf{f}'
\tag{2.2.24}
$$

е еквивалентна на система с изходната матрица A и разредена дясна част, която се решава непълно

$$
A\widetilde{\mathbf{x}} = \begin{pmatrix} \widetilde{\mathbf{f}}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}\,.
\tag{2.2.25}
$$

Наистина, ако (2.2.25) се запише във вида (2.2.20) и се изпълни блочна елиминация се получава точно (2.2.24) и следователно за решенията $\mathbf{x}$ и $\widetilde{\mathbf{x}}$ на изходната система и на (2.2.25) е в сила $\widetilde{\mathbf{x}}_m = \mathbf{x}_m$.

Системата (2.2.25) се решава непълно чрез Алгоритъм SRHS (вж. т. 2.1.2) с $r = 1$ и $d = 1$, т.е. като се търси само една блочна компонента ($\widetilde{\mathbf{x}}_m$) на решението и дясната страна има една ненулева компонента ($\widetilde{\mathbf{f}}_1$).

Стъпките на SM се записват по следния начин:

Алгоритъм SM

Стъпка 1 *Решаване* на системата $U\mathbf{y}_1 = \mathbf{f}'$ чрез стандартния обратен ход;
Пресмятане на дясната страна $\widetilde{\mathbf{f}}_1 = \mathbf{f}_1 - C\mathbf{y}_1$;

Стъпка 2 *Непълно решаване* на системата $A\widetilde{\mathbf{x}} = (\widetilde{\mathbf{f}}_1^T\,, \mathbf{0}^T\,, \ldots\,, \mathbf{0}^T)^T$, само за $\widetilde{\mathbf{x}}_m$ използвайки Алгоритъм SRHS (с $d = 1, j_1 = 1$ и $r = 1, j_1' = m$) с цел получаване на решението $\mathbf{x}_m$ на системата $S\mathbf{x}_m = \widetilde{\mathbf{f}}_1$;

Стъпка 3 *Пресмятане* на дясната страна $\widehat{\mathbf{f}}' = -G\mathbf{x}_m + \mathbf{f}'$;
Решаване на системата $U\mathbf{x}' = \widehat{\mathbf{f}}'$ чрез стандартния обратен ход.

Край {SM}.

За изчислителната сложност на Алгоритъм SM е в сила твърдението:

Теорема 2.2.1 *Марш алгоритъмът, в комбинация с техниката за непълно решаване на редуцираната система, изисква оптимален брой аритметични операции за решаване на задачи $A\mathbf{x} = \mathbf{f}$ с разделящи се променливи, а именно $\mathcal{N}_{SM} \approx 31nm$ операции.*

Доказателство. **Стъпки 1** и **3** изискват $2((m-2)(11n-4)+n)$ ар. оп. за решаване на системите с U (чрез (2.2.23)) и $19n-8$ ар. оп. за пресмятане на десните страни. Съгласно **Теорема 2.1.2**, за непълно решаване на системата в **Стъпка 2** са необходими $m(2(1+1)n+5n-4) = m(9n-4)$ ар. оп. Следователно $\mathcal{N}_{SM} = 2((m-2)(11n-4)+n)+19n-8+m(9n-4) \approx 31nm$. ∎

2.2.2 Обобщен марш алгоритъм

Както е показано от R. Bank и D. Rose в [14, 16], основният недостатък на Алгоритъм SM е, че той е неустойчив за големи m. По-точно, рекурентната връзка, използвана за решаването на системи с горнотриъгълната матрица U от голям блочен ред m, е неустойчива. По тази причина, стандартният марш алгоритъм има практическо приложение само ако дължината на рекурсията е малка, т.е. за $m \ll n$.

Когато $m \approx n$, за задачата (1.1.6) може да се използва обобщения марш алгоритъм (GMA). Нека (за удобство в изложението) $m+1 = p(k+1)$ за някои цели k и p. Системата (1.1.6) най-напред се пренарежда и записва в две-на-две блочна форма

$$\begin{pmatrix} \widetilde{A}_{1,1} & \widetilde{A}_{1,2} \\ \widetilde{A}_{2,1} & \widetilde{A}_{2,2} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^{(1)} \\ \mathbf{f}^{(2)} \end{pmatrix}.$$

След това, прилагайки отново блочна Гаусова елиминация тя се редуцира до решаване на две системи с блока $\widetilde{A}_{1,1}$ и една система с допълнението на Шур $S = \widetilde{A}_{2,2} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\widetilde{A}_{1,2}$. Тук $\widetilde{A} = \left(\widetilde{A}_{i,j}\right)_{i,j=1}^{2}$ е симетрично четно-нечетно пренареждане на A. То е получено чрез едновременно пренареждане на неизвестния вектор $\mathbf{x}^T = (\mathbf{x}^{(1)^T}, \mathbf{x}^{(2)^T})$ и на дясната страна $\mathbf{f}^T = (\mathbf{f}^{(1)^T}, \mathbf{f}^{(2)^T})$ по правилото: всички редове на $\mathbf{x}$ (съответно на $\mathbf{f}$) с номер кратен на $k+1$, образуват втората блочна компонента $\mathbf{x}^{(2)}$. Блокът $\mathbf{x}^{(2)}$ разделя $n \times m$ - мрежата на p ивици, всяка с по k линии и следователно блоковете по диагонала на $\widetilde{A}$ са блочно-диагонални. По-точно, те се дефинират чрез

$$\begin{aligned} \widetilde{A}_{1,1} &= \text{blockdiag}(A_s^{(k)})_{s=1}^{p}, \qquad A_s^{(k)} = I_k \otimes T + B_s^{(k)} \otimes I_n, \\ B_s^{(k)} &= \text{tridiag}(b_{k_s+i,k_s+i-1}\,, b_{k_s+i,k_s+i}\,, b_{k_s+i,k_s+i+1})_{i=1}^{k}\,, \\ \widetilde{A}_{2,2} &= \text{blockdiag}(T + b_{k_{s+1},k_{s+1}}\,I_n),\ k_s = (s-1)(k+1),\ s = 1,\ldots,p\,. \end{aligned}$$

Компонентите $\mathbf{x}^{(i)}, \mathbf{f}^{(i)}, i = 1, 2$ на решението и дясната част се групират и записват в следния блочен вид:

$$\mathbf{x}^{(1)} = \begin{pmatrix} \mathbf{x}_1^{(1)} \\ \vdots \\ \mathbf{x}_p^{(1)} \end{pmatrix}, \ \mathbf{x}_s^{(1)} = \begin{pmatrix} \mathbf{x}_{(s-1)(k+1)+1} \\ \vdots \\ \mathbf{x}_{(s-1)(k+1)+k} \end{pmatrix}, \ s = 1, \ldots, p\,,$$

$$\mathbf{f}^{(1)} = \begin{pmatrix} \mathbf{f}_1^{(1)} \\ \vdots \\ \mathbf{f}_p^{(1)} \end{pmatrix}, \ \mathbf{f}_s^{(1)} = \begin{pmatrix} \mathbf{f}_{(s-1)(k+1)+1} \\ \vdots \\ \mathbf{f}_{(s-1)(k+1)+k} \end{pmatrix}, \ s = 1, \ldots, p\,,$$

$$\mathbf{x}^{(2)^T} = (\mathbf{x}_{k+1}^T, \ldots, \mathbf{x}_{s(k+1)}^T, \ldots, \mathbf{x}_{(p-1)(k+1)}^T),$$
$$\mathbf{f}^{(2)^T} = (\mathbf{f}_{k+1}^T, \ldots, \mathbf{f}_{s(k+1)}^T, \ldots, \mathbf{f}_{(p-1)(k+1)}^T).$$

Чрез блочна факторизация се получава

$$\begin{pmatrix} \widetilde{A}_{1,1} & 0 \\ \widetilde{A}_{2,1} & S \end{pmatrix} \begin{pmatrix} I & \widetilde{A}_{1,1}^{-1}\widetilde{A}_{1,2} \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^{(1)} \\ \mathbf{f}^{(2)} \end{pmatrix},$$

където $S = \widetilde{A}_{2,2} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\widetilde{A}_{1,2}$.

След въвеждане на ново неизвестно

$$\begin{pmatrix} \mathbf{y}^{(1)} \\ \mathbf{y}^{(2)} \end{pmatrix} = \begin{pmatrix} I & \widetilde{A}_{1,1}^{-1}\widetilde{A}_{1,2} \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{pmatrix}$$

и извършване на преобразувания, аналогични на тези в т. 1.2.1 задачата се свежда до решаване на системите

$$\text{(2.2.26)} \qquad \left| \begin{array}{l} \widetilde{A}_{1,1}\mathbf{y}^{(1)} = \mathbf{f}^{(1)} \\ \widetilde{A}_{2,2} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\widetilde{A}_{1,2}\mathbf{x}^{(2)} = \mathbf{f}^{(2)} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\mathbf{f}^{(1)} \\ \widetilde{A}_{1,1}\mathbf{x}^{(1)} = \mathbf{f}^{(1)} - \widetilde{A}_{1,2}\mathbf{x}^{(2)} \end{array} \right. .$$

Тъй като матрицата $\widetilde{A}_{1,1}$ е блочнодиагонална, решаването на система с нея се свежда до решаване на p системи с матрици $A_s^{(k)}$, $s = 1, \ldots, p$, като за всяка от тях се използва стандартния марш алгоритъм.

В този случай, дължината на рекурентната връзка, необходима за решаване на системите с горнотриъгълни блокове е $k-1 = \dfrac{n+1}{p} - 2$ и се контролира чрез избиране на достатъчно голямо p.

Решаването на редуцираната система

$$\text{(2.2.27)} \qquad \widetilde{A}_{2,2} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\widetilde{A}_{1,2}\mathbf{x}^{(2)} = \widetilde{\mathbf{f}}^{(2)} \equiv \mathbf{f}^{(2)} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\mathbf{f}^{(1)}$$

е еквивалентно на непълното решаване на система с първоначалната матрица A и с разредена дясна част. Затова втората стъпка на блочната факторизация е решаването на системата

$$A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}, \text{ където } \widehat{\mathbf{f}}_i = \begin{cases} \widetilde{\mathbf{f}}_s^{(2)}, & i = s(k+1) \\ 0, & i \neq s(k+1) \end{cases}, \tag{2.2.28}$$

търсейки само $\widehat{\mathbf{x}}_{s(k+1)} = \mathbf{x}_{s(k+1)}$, $s = 1, \ldots, p-1$.

Тази задача може да се реши с FASV за $r = p-1$ (брой търсени компоненти на решението) и $d = p-1$ (брой ненулеви компоненти на дясната част), т.е. като се изпълнят последните $\log(p)$ стъпки на правия ход и първите $\log(p)$ на обратния ход. Този подход е предложен от P.S. Vassilevski [94] и съответният алгоритъм се означава в тази работа с GMF. Друга възможност (предложена от G. Bencheva [20]) е да се използва отново Алгоритъм SRHS, като полученият последователен алгоритъм GMS има асимптотично по-голяма изчислителна сложност. Предимствата на GMS са по отношение на паралелните свойства и се разглеждат по-подробно в следващата глава.

При пресметнати $\mathbf{x}^{(2)}$ и $\mathbf{y}^{(1)}$, $\mathbf{x}^{(1)}$ се намира от съответната система в (2.2.26) с матрица $\widetilde{A}_{1,1}$, т.е. чрез решаване на p системи с матрици $A_s^{(k)}$, $s = 1, \ldots, p$.

Обобщеният марш алгоритъм накратко може да бъде записан във вида:

Алгоритъм GMA

Стъпка 1: for $s = 1$ to p

решаване на системите $A_s^{(k)} \mathbf{y}_s^{(1)} = \mathbf{f}_s^{(1)}$ чрез Алгоритъм SM;

end {цикъл по s}

пресмятане на дясната страна $\widetilde{\mathbf{f}}^{(2)} = \mathbf{f}^{(2)} - \widetilde{A}_{2,1}\, \mathbf{y}^{(1)}$;

Стъпка 2: *непълно решаване* на системата

$$A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}, \text{ където } \widehat{\mathbf{f}}_i = \begin{cases} \widetilde{\mathbf{f}}_s^{(2)}, & i = s(k+1) \\ 0, & i \neq s(k+1) \end{cases},$$

намирайки само $\widehat{\mathbf{x}}_{s(k+1)} = \mathbf{x}_{s(k+1)}$, $s = 1, \ldots, p-1$,

с цел намиране на компонентата $\mathbf{x}^{(2)}$;

Стъпка 3: *пресмятане* на дясната страна $\widetilde{\mathbf{f}}^{(1)} = \mathbf{f}^{(1)} - \widetilde{A}_{1,2}\, \mathbf{x}^{(2)}$;

for $s = 1$ to p

решаване на системите $A_s^{(k)} \mathbf{x}_s^{(1)} = \widetilde{\mathbf{f}}_s^{(1)}$ чрез Алгоритъм SM;

end{цикъл по s}

Край {GMA}.

Предполага се за определеност, че $p = 2^{l_p}$. Изчислителната сложност на Алгоритъм GMA зависи от начина на решаване на задачата в **Стъпка 2**.

Теорема 2.2.2 *Обобщеният марш алгоритъм в комбинация с бързия алгоритъм за разделяне на променливите и техниката за непълно решаване на задачи с разредена дясна част, т.е.* Алгоритъм GMF, *изисква* $\mathcal{N}_{GMF} \approx 62pkn+24nm(\log(p)-1)-9nm$ *ар. оп. за решаване на задача (1.1.6). Неговата модификация GMS използваща само техниката за непълно решаване на задачи с разредена дясна част изисква* $\mathcal{N}_{GMS} \approx 62pkn + 4pnm + nm$ *ар. оп.*

Доказателство. **Стъпки 1** и **3** на GMA (и двата варианта) представляват решаване на p системи от блочен ред k чрез Алгоритъм SM. От **Теорема 2.2.1** следва, че за тях са необходими $2p(31kn-25n-12k+8)$ ар. оп., а пресмятането на съответните десни страни се извършва за $8n(p-1)$ ар. оп. За **Стъпка 2** на GMF, т.е когато се използва FASV за непълно решаване на (2.2.28), трябва да се извършат $\log(p)$ стъпки на FASV и от **Теорема 2.1.3** следва, че тук са необходими $24nm(\log(p)-1)-9nm$ ар. оп. Когато се използва SRHS, от **Теорема 2.1.2** с $r = d = p-1$ следва, че втората стъпка на GMS изисква $4pnm + nm$ ар. оп. С това теоремата е доказана. ∎

Забележка: Обобщеният марш алгоритъм във вида предложен от R. Bank [14] изисква за втората стъпка на блочната Гаусова елиминация $28n^2\log(p)$ операции. Това показва, че алгоритъм GMF представен по-горе има асимптотично по-малък брой операции.

2.3 Числени експерименти

Описаните до тук алгоритми SV, FASV, SM, GMF и GMS са реализирани както на C, така и на Fortran77. В тази секция се представят резултати от числени експерименти със C-кодовете върху Power Macintosh компютър с 512MB RAM и процесор Power PC G4/450MHz. Резултати от кодовете на Fortran77 върху компютър HP9000/C110 са публикувани в G. Bencheva [17] (за SV, FASV) и в G. Bencheva [18] (за SM, GMF).

2.3.1 Тестови примери

За експериментално изследване на устойчивостта и изчислителната ефективност на разглежданите алгоритми са използвани следните два тестови примера.

Пример 1. *Приложение за случая* $a_1(y_1) = a_2(y_2) \equiv 1$.

Разглежда се задачата на Дирихле за уравнението на Поасон :

$$\left|\begin{array}{rcll} -\Delta u(y_1,y_2) & = & f(y_1,y_2), & (y_1,y_2)\in\Omega=(0,1)\times(0,1) \\ u(y_1,y_2) & = & 0, & (y_1,y_2)\in\partial\Omega \end{array}\right. .$$

За решение се избира $u(y_1,y_2)=\sin(\pi y_1)\sin(\pi y_2)$ *откъдето следва, че дясната част е* $f(y_1,y_2)=2\pi^2\sin(\pi y_1)\sin(\pi y_2)$.

Пример 2. *Приложение за случая* $a_1(y_1)\neq 1$ *и* $a_2(y_2)\neq 1$.

Разглежда се задача (1.1.2), с коефициенти $a_1(y_1)=1+y_1^2$, $a_2(y_2)=e^{-y_2}$ *и решение* $u(y_1,y_2)=(1-y_1)y_1y_2(1-y_2)$. *Следователно дясната страна е* $f(y_1,y_2)=2y_2(1-y_2)(3y_1^2-y_1+1)+e^{-y_2}y_1(1-y_1)(3-2y_2)$.

И в двата случая дискретизацията е направена върху равномерна мрежа $n\times n$ ($m=n$, $h_1=h_2=h$) с възли $(y_1^i,y_2^j)=(i\,h,j\,h)$, $i,j=1,\ldots,n$, $h=\dfrac{1}{n+1}$ с помощта на диференчна апроксимация по петточков шаблон „кръст“.

2.3.2 Устойчивост на алгоритмите

Като мярка за устойчивостта на изследваните алгоритми е използвано поведението на грешката в дискретна l_2- и C-норма. За даден вектор $\mathbf{v}=(v_1,\ldots,v_n)$, те се дефинират съответно чрез $\|\mathbf{v}\|_{l_2}=\sqrt{\dfrac{1}{n}\sum_{i=1}^{n}v_i^2}$ и $\|\mathbf{v}\|_C=\max\limits_{1\le i\le n}|v_i|$.

В първата колона на всички таблици е даден параметърът n като характеристика на мрежовата стъпка h. Останалите колони са в групи по 2 – за l_2- и C-нормите на грешката. Ако е необходимо допълнително уточняване на параметрите на задачата, колоните са в групи по 3, като първата е за параметъра, а другите 2 – за грешката.

Таблица 2.1: Грешка за SV и FASV.

	Пример 1.		**Пример 2.**	
n	l_2	C	l_2	C
15	1.609e-3	3.219e-3	2.159e-5	4.107e-5
31	4.018e-4	8.036e-4	5.396e-6	1.029e-5
63	1.004e-4	2.008e-4	1.349e-6	2.573e-6
127	2.510e-5	5.020e-5	3.372e-7	6.434e-7
255	6.275e-6	1.255e-5	8.431e-8	1.608e-7
511	1.569e-6	3.137e-6	2.108e-8	4.021e-8
1023	3.922e-7	7.844e-7	5.270e-9	1.005e-8

Таблица 2.2: Грешка за SM.

	Пример 1		**Пример 2**	
n	l_2	C	l_2	C
3	2.651e-2	5.303e-2	3.453e-4	6.513e-4
7	6.475e-3	1.295e-2	8.639e-5	1.624e-4
15	1.609e-3	3.219e-3	1.788e+9	1.712e+10
31	6.086e+8	5.611e+9	1.527e+44	2.577e+45

Получените норми на грешката за алгоритмите SV и FASV за различни стойности на мрежовия параметър n са съответно равни и са представени в Таблица 2.1 за двата тестови примера. При намаляване на стъпката два пъти и двете норми намаляват приблизително четири пъти. Това е така, защото методите са директни и получената грешка е в същност грешката на диференчната схема, т.е. $\mathcal{O}\left(|h|^2\right)$ (вж. т. 1.1). Получените резултати потвърждават, че алгоритмите SV и FASV са устойчиви.

Резултатите, представени в Таблица 2.2 показват, че алгоритъм SM е „условно“ устойчив при $m \leq 15$ за **Пример 1**, и при $m \leq 7$ за **Пример 2**.

В Таблици 2.3 и 2.4 са дадени резултатите за GMF и GMS съответно за **Пример 1** и **Пример 2**. Освен n, тук се променя и размерът на стъпката $k-1$ на рекурсията в

Таблица 2.3: Грешка за GMF и GMS ($n = p(k+1) - 1$), **Пример 1.**

	$k=3$			$k=15$				
					GMF		GMS	
n	p	l_2	C	p	l_2	C	l_2	C
7	2	6.475e-3	1.295e-2					
15	4	1.609e-3	3.219e-3					
31	8	4.018e-4	8.036e-4	2	4.307e-4	1.634e-3	4.307e-4	1.634e-3
63	16	1.004e-4	2.008e-4	4	1.180e-4	1.090e-3	1.116e-4	9.616e-4
127	32	2.510e-5	5.020e-5	8	2.872e-5	3.215e-4	2.805e-5	2.822e-4
255	64	6.275e-6	1.255e-5	16	6.975e-6	7.863e-5	6.974e-6	6.845e-5
511	128	1.569e-6	3.137e-6	32	1.837e-6	3.595e-5	1.845e-6	3.029e-5
1023	256	3.922e-7	7.844e-7	64	7.282e-7	1.796e-5	7.320e-7	1.757e-5

GMF и GMS. Стойностите на k се избират така, че алгоритъм SM да е устойчив. За $k = 3$ получените норми на грешката съвпадат за GMF и GMS и за двата примера.

Таблица 2.4: Грешка за GMF и GMS ($n = p(k+1) - 1$), **Пример 2.**

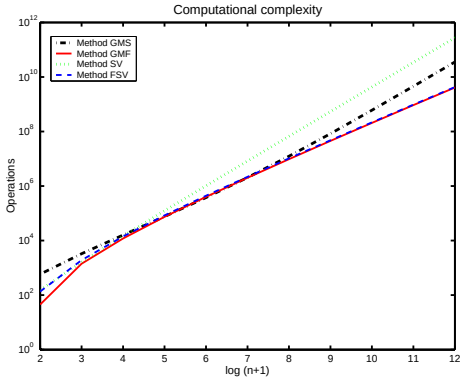
	$k = 3$			$k = 7$		
n	p	l_2	C	p	l_2	C
7	2	8.639e-5	1.624e-4			
15	4	2.159e-5	4.107e-5	2	2.159e-5	4.107e-5
31	8	5.396e-6	1.029e-5	4	5.396e-6	1.029e-5
63	16	1.349e-6	2.573e-6	8	1.349e-6	2.573e-6
127	32	3.372e-7	6.434e-7	16	3.372e-7	6.434e-7
255	64	8.431e-8	1.608e-7	32	8.431e-8	1.608e-7
511	128	2.108e-8	4.021e-8	64	2.108e-8	4.021e-8
1023	256	5.270e-9	1.005e-8	128	5.270e-9	1.007e-8

Поведението им е като това на SV и FASV, а именно при двукратно намаляване на стъпката и двете норми намаляват приблизително 4 пъти. Резултатите за $k = 7$ са подобни (за **Пример 1** не са показани в таблиците) с едно изключение – за **Пример 2** при $n = 1023$ C-нормата за $k = 7$ на GMS е $1.006e - 8$. Следователно GMF и GMS са устойчиви за $k = 3$ и $k = 7$ и за двата тестови примера. При $k = 15$, SM е устойчив само за **Пример 1**. Независимо от това, устойчивостта на GMF и GMS в този случай е нарушена, като резултатите са все още приемливи. Това означава, че на практика в зависимост от специфичната аритметика с плаваща запетая, трябва много внимателно да се избира стойността на параметъра k, т.е. той зависи допълнително от конкретната машина и компилатора.

2.3.3 Сравнителен анализ на изчислителната ефективност

Времето за изпълнение на разглежданите методи зависи от асимптотиката на тяхната изчислителна сложност, посочена в съответните теореми на т. 2.1 и т. 2.2. На Фиг. 2.1 са представени аналитично и с графика (за $k = 7$) водещите им членове за случая $n = m = 2^l - 1$, $n = p(k+1) - 1$. Алгоритъм SM е най-бърз, но не е изобразен на графиката, тъй като за големи n не е устойчив. За малки стойности на l, т.е. за $l < 4$, броят операции за изпълнение на FASV е по-голям от този за изпълнение на SV. Графиката на изчислителната сложност на GMF е изцяло под тази на FASV, когато $k = 7$ или 15, а за $k = 3$ местата им са разменени. Отново за малки стойности на l, броят операции за изпълнение на GMS е по-малък от този за изпълнение на GMF. В зависимост от стойността на k диапазонът им леко се изменя: при $k = 3$

Метод	Брой операции
SV	$4n^3 + 5n^2$
FASV	$24n^2 \log_2(n+1) - 33n^2$
SM	$31n^2$
GMF	$24n^2 \log_2(p) + 29n^2$
GMS	$4pn^2 + 63n^2$



Фиг. 2.1: Методи за разделяне на променливите – изчислителна сложност.

интервалът е $3 < l < 6$; за $k = 7$ той е $4 < l < 8$ и за $k = 15$ е $4 < l < 9$. С други думи, асимптотично GMF е най-бърз, следван от FASV, GMS и SV, но съществуват интервали от стойности за l, в които поне един от „бавните“ алгоритми GMS и SV е по-бърз от някой от „бързите“.

В следващите таблици са дадени измерените процесорни времена за изпълнение на всеки от алгоритмите SV, FASV, GMF и GMS за двата примера, като не е включено времето за намиране на необходимите собствени стойности и вектори.

Таблица 2.5: Времена за SV и FASV.

	Пример 1.		**Пример 2.**	
n	SV	FASV	SV	FASV
63	0.03	0.23	0.03	0.21
127	0.26	0.87	0.29	0.83
255	3.14	3.86	3.41	3.77
511	33.71	16.88	33.82	16.66
1023	285.38	70.38	284.73	69.70

Времето за изпълнение на всеки от алгоритмите с увеличаване на n се изменя съобразно изразите, дадени на Фиг. 2.1. С увеличаване на k, времената за GMF и GMS намаляват и са по-малки от това на FASV. В същото време интервалите за l, в които „бавните“ алгоритми SV и GMS са по-бързи от „бързите“ FASV и GMF са изместени. По-точно, FASV отнема по-малко време от SV едва при $n = 511$ (т.е. $l = 9$); GMF е по-бърз от GMS при $n = 1023$ за $k = 3$ и $k = 7$. За достатъчно големи задачи „бързите“ според теоретичния анализ алгоритми се изпълняват за по-малко време. За важни за практическите приложения размерности се оказва, че GMS може да е по-бърз от тях и не бива да се пренебрегва.

Таблица 2.6: Времена за GMF и GMS ($n = p(k+1) - 1$), **Пример 1.**

	$k = 3$		$k = 7$		$k = 15$	
n	GMF	GMS	GMF	GMS	GMF	GMS
63	0.20	0.10	0.11	0.07	0.07	0.05
127	0.89	0.49	0.47	0.29	0.31	0.18
255	3.84	2.47	2.25	1.43	1.37	0.81
511	16.69	15.63	9.97	8.59	6.41	4.42
1023	70.30	101.66	42.89	52.03	28.13	25.04

Таблица 2.7: Времена за GMF и GMS ($n = p(k+1) - 1$), **Пример 2.**

	$k = 3$		$k = 7$	
n	GMF	GMS	GMF	GMS
63	0.21	0.11	0.08	0.06
127	0.88	0.47	0.52	0.28
255	3.82	2.65	2.23	1.39
511	16.44	15.70	9.84	8.40
1023	69.45	100.64	42.04	51.39

Тъй като най-често решаването на елиптични задачи с разделящи се променливи се използва на някои от стъпките за решаването на по-сложни задачи, факторът време за намиране на решението е от особено значение. Представените резултати показват, че освен изчислителната сложност, върху него влияят и фактори като структура на алгоритъма и средства на компилатора (напр. условни оператори, вложени цикли, извикване на функции, включително и за заделяне и освобождаване на памет – вж. напр. R. Heathfield, L. Kirby et al. [57] за времената на някои от тях спрямо аритметичните операции). Като се вземат под внимание и предимствата му по отношение на паралелна реализация (вж. Глава 3), GMS дава сериозни основания да се разглежда, заедно с другите бързи методи при конструирането на ефективни алгоритми (в частност и преобусловители) за решаването на по-сложни задачи.

2.4 Приложение при конструиране на преобусловители

Освен самостоятелното значение за решаване на елиптични задачи с разделящи се променливи в правоъгълна област, изследваните до тук преки алгоритми имат при-

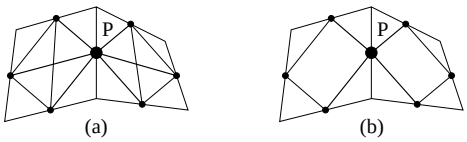
ложение и при конструирането на ефективни преобусловители за задачи в по-общ вид като: а) задачи с плавно променящи се коефициенти и/или скокове на коефициентите съответстващи на случая на многослойна среда; б) съставни, напр. L-образни или T-образни, области (вж. напр. [69, 84, 95]).

В тази секция, чрез локална модификация на елементните матрици на коравина, се конструира ефективен преобусловител с разделящи се променливи за елиптична задача от втори ред дискретизирана с неконформни четириъгълни крайни елементи (G. Bencheva, S. Margenov [23]). За решаването ѝ се използва метод на спрегнатия градиент с преобуславяне. Системата с преобусловителя се решава чрез Алгоритъм FASV.

2.4.1 Същност на алгоритъма

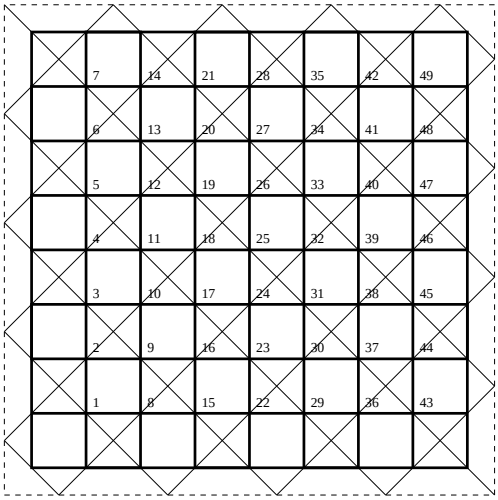
Разглежда се задача (1.1.1) в случая на хомогенни гранични условия на Дирихле $\mu(y) \equiv 0$, дискретизирана чрез неконформни четириъгълни КЕ. При стандартна равномерна мрежа и номерация на възлите (Фигура 1.3а)), дискретната система има разредена матрица с не повече от 7 ненулеви елемента на ред (Фигура 1.3б)). Структурата ѝ е специфична, различна от блочно-лентовия случай при конформни елементи, което затруднява директното прилагане на много от съществуващите бързи преки алгоритми. Изборът на подходящ преобусловител е от съществено значение за ефективното решаване на системата чрез PCG (виж т. 1.2.2).

Всеки възел P от мрежата има връзки (в смисъл на шаблона на апроксимация) с възлите от два съседни четириъгълни елемента, както е показано на Фигура 2.2(a).



Фиг. 2.2: Схема на свързаност.

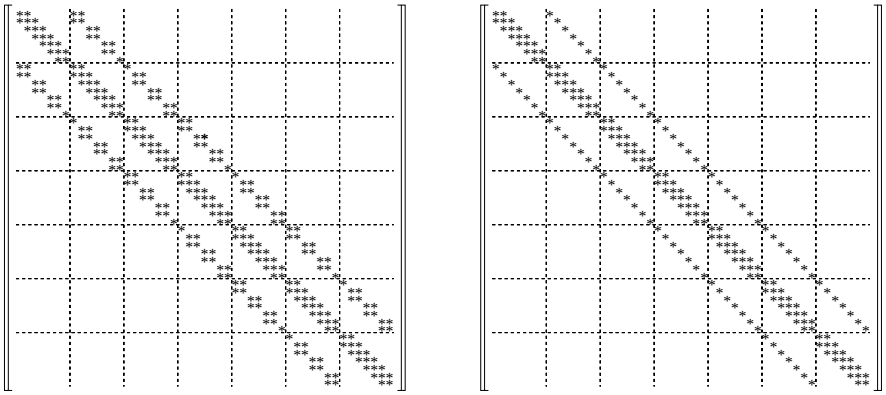
Ако се „отрежат“ две от връзките, се получава схема на свързване подобна на наклонен петточков шаблон „кръст“ (Фигура 2.2(b)). Използването като преобусловител на система, получена след дискретизация чрез петточков шаблон, се мотивира от множеството съществуващи бързи алгоритми за решаването ѝ. Стандартното подреждане и номерация на възлите за неконформни КЕ не води до необходимата блочно-тридиагонална структура за преобусловителя. За да се преодолее този проблем се завърта мрежата (Фигура 2.3). Избира се наклонена мрежа за неконформния МКЕ и нормално ориентирана – за спомагателната задача. Стандартната номерация на



Фиг. 2.3: *Наклонена* мрежа за неконформни КЕ и *нормално* ориентирана мрежа за спомагателната задача.

възлите следва нормално ориентираната мрежа (вж. също и Фигура 2.6).

Матрицата A на системата за това подреждане има регулярната структура показана на Фигура 2.4(а). Нека N_i е броят неконформни елементи по направление y_i на мрежата ω_h. Тогава размерността на A след налагане на граничните условия е $N = (2\,N_1 - 1)(2\,N_2 - 1)$. Преобуславящата матрица B има добре известната блочно-тридиагонална структура с диагонални извъндиагонални блокове и тридиагонални диагонални блокове (вж. Фигура 2.4(б)).



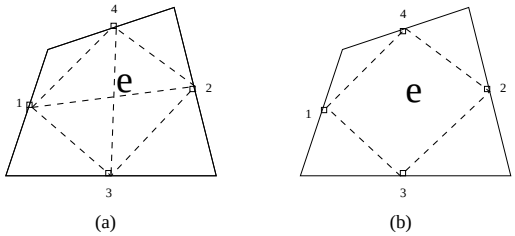
(а) (б)

Фиг. 2.4: Структура на матрицата на коравина (а) и на преобусловителя (б).

Тези две матрици са получени чрез стандартната процедура на асемблиране при МКЕ от елементните матрици на коравина A_e и B_e:

$$A = \sum_{e \in \omega_h} L_e^t\, A_e\, L_e\,, \qquad B = \sum_{e \in \omega_h} L_e^t\, B_e\, L_e\,.$$

Тук L_e представлява рестрикцията на глобалния вектор на неизвестните върху e. Локалната номерация на възлите за елементите e е представена на Фигура 2.5(а),(б). Матриците A_e и B_e са положително полуопределени, като A_e са плътни за всяко e,



Фиг. 2.5: Локална номерация на възлите (а) за елементната матрица A_e и (б) за модифицираната матрица B_e.

а B_e имат по 4 нулеви елемента съответстващи на „отрязаните“ връзки:

$$A_e = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}, \quad B_e = \begin{bmatrix} \beta_{11} & 0 & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ \beta_{31} & \beta_{32} & \beta_{33} & 0 \\ \beta_{41} & \beta_{42} & 0 & \beta_{44} \end{bmatrix}. \tag{2.4.29}$$

За ненулевите компоненти на B_e е в сила $\beta_{ij} = \beta_{ji}$, $\beta_{ij} = \alpha_{ij}$, $i = 1, 2$, $j = 3, 4$ и

$$\beta_{11} = \alpha_{11} + \alpha_{12}\,,\ \beta_{22} = \alpha_{22} + \alpha_{21}\,,\ \beta_{33} = \alpha_{33} + \alpha_{34}\,,\ \beta_{44} = \alpha_{44} + \alpha_{43}\,.$$

Структурата на B съответства на обобщен наклонен петточков шаблон „кръст“. Това означава, че за широк клас дискретизации, B може да се разглежда като матрица получена чрез МКР или линейни конформни КЕ. Накрая се прилага скалиране на коефициентите на елементните матрици на коравина за B, за да се получи задача с разделящи се променливи.

2.4.2 Моделен анализ на числото на обусловеност

Анализира се анизотропна задача асоциирана с билинейната форма

$$a_h(u, v) = \sum_{e \in \omega_h} \int_e a(e)\left(\frac{\partial u}{\partial y_1}\frac{\partial v}{\partial y_1} + \varepsilon \frac{\partial u}{\partial y_2}\frac{\partial v}{\partial y_2}\right) dy\,.$$

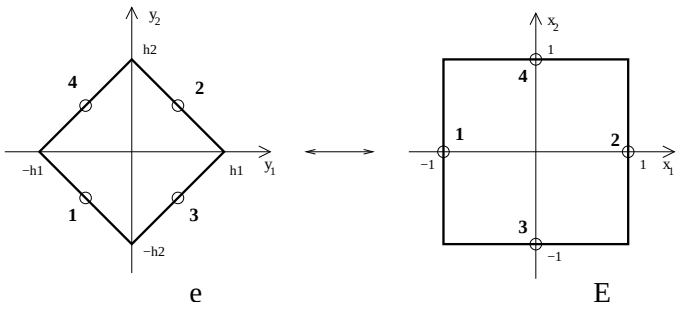
Тук ω_h е завъртяна мрежа, състояща се от квадратни елементи с диагонали, успоредни на координатните оси. Коефициентът $a(e)$ е константа върху всеки елемент $e \in \omega_h$, а $\varepsilon > 0$ е параметърът на анизотропия.

Елементните матрици на коравина съответстващи на MP и MV ротирани билинейни елементи (описани в т. 1.1.2) са

$$A_e^{MP} = \frac{a(e)}{6}\begin{bmatrix} 5(1+\varepsilon) & -1-\varepsilon & -5+\varepsilon & 1-5\varepsilon \\ -1-\varepsilon & 5(1+\varepsilon) & 1-5\varepsilon & -5+\varepsilon \\ -5+\varepsilon & 1-5\varepsilon & 5(1+\varepsilon) & -1-\varepsilon \\ 1-5\varepsilon & -5+\varepsilon & -1-\varepsilon & 5(1+\varepsilon) \end{bmatrix},$$

$$A_e^{MV} = \frac{a(e)}{4}\begin{bmatrix} 5(1+\varepsilon) & 1+\varepsilon & -5-\varepsilon & -1-5\varepsilon \\ 1+\varepsilon & 5(1+\varepsilon) & -1-5\varepsilon & -5-\varepsilon \\ -5-\varepsilon & -1-5\varepsilon & 5(1+\varepsilon) & 1+\varepsilon \\ -1-5\varepsilon & -5-\varepsilon & 1+\varepsilon & 5(1+\varepsilon) \end{bmatrix}.$$

Те са пресметнати по стандартната схема на МКЕ с помощта на възловите базисни



Фиг. 2.6: Локален елемент e и базов елемент E.

функции $\widehat{\phi}_i$, $i = 1, \ldots, 4$ (вж. т. 1.1.2) и смяна на променливите изобразяваща базовия елемент E в $e \in \omega_h$ (Фигура 2.6).

Локалните матрици B_e се получават като приложим (2.4.29) и имат вида:

$$B_e^{MP} = \frac{a(e)}{6}\begin{bmatrix} 4(1+\varepsilon) & 0 & -5+\varepsilon & 1-5\varepsilon \\ 0 & 4(1+\varepsilon) & 1-5\varepsilon & -5+\varepsilon \\ -5+\varepsilon & 1-5\varepsilon & 4(1+\varepsilon) & 0 \\ 1-5\varepsilon & -5+\varepsilon & 0 & 4(1+\varepsilon) \end{bmatrix},$$

$$B_e^{MV} = \frac{a(e)}{4}\begin{bmatrix} 6(1+\varepsilon) & 0 & -5-\varepsilon & -1-5\varepsilon \\ 0 & 6(1+\varepsilon) & -1-5\varepsilon & -5-\varepsilon \\ -5-\varepsilon & -1-5\varepsilon & 6(1+\varepsilon) & 0 \\ -1-5\varepsilon & -5-\varepsilon & 0 & 6(1+\varepsilon) \end{bmatrix}.$$

За да оценим числото на обусловеност $\kappa(B^{-1}A)$ прилагаме локален анализ за A_e и B_e. Матрицата B_e^{MP} е положително полуопределена при $\varepsilon \in [\frac{1}{5}, 5]$, а за A_e^{MP}, A_e^{MV}

и B_e^{MV} това е в сила за всяко $\varepsilon > 0$. Освен това, за $\varepsilon \in (\frac{1}{5}, 5)$ имаме $Ker(A_e^{MP}) = Ker(B_e^{MP}) = Span\{\mathbf{e}\}$, където $\mathbf{e}^t = (1,1,1,1)$, а $Ker(A_e^{MV}) = Ker(B_e^{MV}) = Span\{\mathbf{e}\}$ за $\varepsilon > 0$. При тези условия локалните спектрални задачи се записват във вида

$$A_e^{MP}\mathbf{w} = \lambda B_e^{MP}\mathbf{w}\,, \quad A_e^{MV}\mathbf{w} = \lambda B_e^{MV}\mathbf{w}\,, \quad \mathbf{w} \neq c\,\mathbf{e}^t\,. \tag{2.4.30}$$

Вместо (2.4.30) е достатъчно да решим редуцирани 3×3 спектрални задачи без ограничения за $\mathbf{w}$ (вж. V. Eijkhout, P. Vassilevski [43]). Да разгледаме случаите $\frac{1}{5} < \varepsilon \leq 1$ за MP и $0 < \varepsilon \leq 1$ за MV. Оценките при $\varepsilon > 1$ са подобни, но с $\frac{1}{\varepsilon}$ вместо с ε.

За MP при $\varepsilon \in (\frac{1}{5}, 1]$ редуцираната спектрална задача води до следното характеристично уравнение за λ:

$$det\begin{bmatrix} (5-4\lambda)(1+\varepsilon) & -1-\varepsilon & (1-\lambda)(\varepsilon-5) \\ -1-\varepsilon & (5-4\lambda)(1+\varepsilon) & (1-\lambda)(1-5\varepsilon) \\ (1-\lambda)(\varepsilon-5) & (1-\lambda)(1-5\varepsilon) & (5-4\lambda)(1+\varepsilon) \end{bmatrix} = 0. \tag{2.4.31}$$

Корените му са $\lambda_1 = 1$, $\lambda_2 = \dfrac{6}{5-\varepsilon}$, $\lambda_3 = \dfrac{6\varepsilon}{5\varepsilon - 1}$ и в разглеждания интервал по отношение на ε са в сила неравенствата $\lambda_1 < \lambda_2 \leq \lambda_3$. При $\varepsilon = 1$, т.е. за изотропна задача, $\lambda_2 = \lambda_3 = \dfrac{3}{2}$ откъдето следва, че $\kappa((B_e^{MP})^{-1}A_e^{MP}) = \dfrac{\lambda_{max}}{\lambda_{min}} = \dfrac{3}{2}$. За $\varepsilon \neq 1$ $\lambda_1 < \lambda_2 < \lambda_3$ и от тук $\kappa((B_e^{MP})^{-1}A_e^{MP}) = \dfrac{\lambda_{max}}{\lambda_{min}} = \dfrac{6\varepsilon}{5\varepsilon - 1}$.

Характеристичното уравнение за MV ($\varepsilon \in (0,1]$)

$$det\begin{bmatrix} (5-6\lambda)(1+\varepsilon) & 1+\varepsilon & (1-\lambda)(-\varepsilon-5) \\ 1+\varepsilon & (5-6\lambda)(1+\varepsilon) & (1-\lambda)(-1-5\varepsilon) \\ (1-\lambda)(-\varepsilon-5) & (1-\lambda)(-1-5\varepsilon) & (5-6\lambda)(1+\varepsilon) \end{bmatrix} = 0 \tag{2.4.32}$$

има корени $\lambda_1 = 1$, $\lambda_2 = \dfrac{4}{5+\varepsilon}$, $\lambda_3 = \dfrac{4\varepsilon}{5\varepsilon+1}$. За $\varepsilon = 1$ се получава $\lambda_1 > \lambda_2 = \lambda_3 = \dfrac{2}{3}$ и $\kappa((B_e^{MV})^{-1}A_e^{MV}) = \dfrac{\lambda_{max}}{\lambda_{min}} = \dfrac{3}{2}$. За $\varepsilon \in (0,1)$ е в сила $\lambda_1 > \lambda_2 > \lambda_3$ и $\kappa((B_e^{MV})^{-1}A_e^{MV}) = \dfrac{\lambda_{max}}{\lambda_{min}} = \dfrac{5\varepsilon+1}{4\varepsilon}$.

Глобалната оценка за числото на обусловеност се получава от представения локален анализ. За всеки от вариантите MP и MV, тя следва директно от неравенствата

$$\mathbf{v}^T A\mathbf{v} = \sum_{e\in\omega_h} \mathbf{v}_e^T L_e^T A_e L_e \mathbf{v}_e \leq \lambda_{max} \sum_{e\in\omega_h} \mathbf{v}_e^T L_e^T B_e L_e \mathbf{v}_e = \lambda_{max}\mathbf{v}^T B\mathbf{v}$$

и аналогично

$$\mathbf{v}^T A\mathbf{v} \geq \lambda_{min}\mathbf{v}^T B\mathbf{v}.$$

Следователно $\kappa(B^{-1}A) \leq \kappa(B_e^{-1}A_e)$ и по-точно

$$\kappa((B^{MP})^{-1}A^{MP}) = \kappa((B^{MV})^{-1}A^{MV}) \leq \frac{3}{2}\,, \qquad \varepsilon = 1\,,$$
$$\kappa((B^{MP})^{-1}A^{MP}) \leq \frac{6\varepsilon}{5\varepsilon - 1}\,, \quad \varepsilon \in (\frac{1}{5}, 1]\,,$$
$$\kappa((B^{MV})^{-1}A^{MV}) \leq \frac{5\varepsilon + 1}{4\varepsilon}\,, \quad \varepsilon \in (0, 1]\,.$$

Това означава, че скоростта на сходимост на PCG не зависи от мрежовия параметър, но зависи от параметъра на анизотропия. Когато $\varepsilon \to \frac{1}{5}$ за MP и $\varepsilon \to 0$ за MV, съответните относителни числа на обусловеност клонят към безкрайност. От друга страна, колкото по-близко е ε до 1, толкова по-бърза е сходимостта.

Изчислителната сложност на една итерация на предложения алгоритъм е

$$\mathcal{N}_{it}^{PCG/FASV}(A^{-1}\mathbf{b}) \approx \mathcal{N}(B^{-1}\mathbf{v}) + \mathcal{N}(A\mathbf{v}) + 10N\,.$$

Матрицата A има не повече от 7 ненулеви елемента на ред и следователно $\mathcal{N}(A\mathbf{v}) \approx 13N$. Системата с преобусловителя B се решава чрез Алгоритъм FASV. Съгласно **Теорема 2.1.3**, $\mathcal{N}(B^{-1}\mathbf{v}) \approx 24nm\log(m+1)$, където $n = 2N_1 - 1$, $m = 2N_2 - 1$ и $N = nm$. Следователно

$$\mathcal{N}_{it}^{PCG/FASV}(A^{-1}\mathbf{b}) \approx 24\,N\,\log\sqrt{N} + 23\,N\,. \tag{2.4.33}$$

От тук, като се отчете, че $\kappa(B^{-1}A)$ не зависи от N следва, че предложеният преобусловител е почти оптимален.

2.4.3 Числени експерименти

Разглежда се гранична задача на Дирихле в единичния квадрат, асоциирана с

$$a_h(u,v) = \sum_{e\in\omega_h}\int_e \sum_{i=1}^{2} a_i(y_1,y_2)\frac{\partial u}{\partial y_i}\frac{\partial v}{\partial y_i}dy\,,\ \Omega = (0,1)^2\,.$$

За получаването на дискретната система за неконформни КЕ се използва наклонена мрежа. Въвежда се спомагателна задача с разделящи се променливи, дефинирана чрез билинейната форма

$$a_h^*(u,v) = \int_{\omega_h}\left(b_1(y_1)\frac{\partial u}{\partial y_1}\frac{\partial v}{\partial y_1} + b_2(y_2)\frac{\partial u}{\partial y_2}\frac{\partial v}{\partial y_2}\right)dy\,, \tag{2.4.34}$$

където коефициентите $b_i(y_i)$, $i = 1, 2$ се определят чрез

$$b_1(y_1) = \int_0^1 c_1 a_1(y_1,y_2) + c_2 a_2(y_1,y_2)\,dy_2\,,$$
$$b_2(y_2) = \int_0^1 c_2 a_1(y_1,y_2) + c_1 a_2(y_1,y_2)\,dy_1\,.$$

Константите c_1 и c_2 се избират така, че да се получи дефинирания по-горе преобусловител. За MP те са $c_1 = 5$, $c_2 = -1$, а за MV са $c_1 = 5$, $c_2 = 1$. Спомагателната задача (2.4.34) се дискретизира чрез МКР върху нормално ориентирана равномерна мрежа. Съответната система се решава чрез FASV.

По-долу се представят резултати за следните четири тестови примера:

Пример 1. $a_1(y_1, y_2) \equiv 1$, $a_2(y_1, y_2) \equiv \varepsilon$;

Пример 2. $a_1(y_1, y_2) = a_2(y_1, y_2) = 1 + \varepsilon \exp(y_1 + y_2)$;

Пример 3. $a_1(y_1, y_2) = a_2(y_1, y_2) = 1 + \dfrac{\varepsilon}{2} \sin(2\pi(y_1 + y_2))$;

Пример 4. $a_1(y_1, y_2) = 1 + \varepsilon \exp(y_1 + y_2)$, $a_2(y_1, y_2) = 1 + \dfrac{\varepsilon}{2} \sin(2\pi(y_1 + y_2))$.

Първият пример е анизотропната задача с постоянни коефициенти от моделния анализ, а другите три са с променливи коефициенти. Вторият пример е представител на задачите с плавно променящи се монотонни коефициенти, а третият е с осцилиращи коефициенти. **Пример 4** е взет от R. Chan, T. Chan [33] и е използван като моделна тестова задача в множество статии излезли от печат след това.

Критерият за спиране на итерационния процес е $\dfrac{(B^{-1}r^{N_{it}}, r^{N_{it}})}{(B^{-1}r^0, r^0)} < 10^{-6}$, където с B е означен преобусловителят, а с r^j – резидуалът на j-тата стъпка на PCG. Като мярка за скоростта на сходимост, в таблична форма се представя броят на итерациите N_{it} за променящ се мрежов параметър $h = \dfrac{1}{n+1}$ и набор от стойности на ε. И в двете таблици * означава, че съответният преобусловител не е положително определен.

Таблица 2.8: Брой на итерациите на PCG за **Пример 1**.

	MP				MV			
$\varepsilon \setminus n$	15	31	63	127	15	31	63	127
1.0	6	6	5	5	6	6	5	5
0.5	14	14	14	13	10	10	10	10
0.25	37	48	48	48	17	17	17	17
0.21	55	94	116	122	19	19	19	19
0.1	*	*	*	*	30	31	31	31
0.01	*	*	*	*	63	98	109	116

За **Пример 1.** (Таблица 2.8), случаят $\varepsilon = 1$ е уравнението на Поасон. Вижда се, че скоростта на сходимост не зависи от размерността на дискретната задача. Представените резултати илюстрират теоретичните оценки, а именно колкото по-близо е ε до $\dfrac{1}{5}$ за MP и до 0 за MV, толкова по-голям е броят на итерациите.

Таблица 2.9 за останалите три примера е с подобен формат, но тук ε не е пара-

Таблица 2.9: Брой на итерациите на PCG.

	Пример 2.				**Пример 3.**				**Пример 4.**			
$n \setminus \varepsilon$	0.0	0.01	0.1	1.0	0.0	0.01	0.1	1.0	0.0	0.01	0.1	1.0
	MP											
15	6	6	7	9	6	6	6	12	6	6	9	*
31	6	6	7	9	6	6	6	13	6	6	10	*
63	5	5	7	9	5	5	6	13	5	6	10	*
127	5	5	7	9	5	5	6	12	5	6	10	*
255	4	5	6	9	4	5	6	12	4	5	10	*
	MV											
15	6	6	7	9	6	6	7	13	6	6	8	24
31	6	6	7	9	6	6	6	13	6	6	8	31
63	5	5	7	9	5	5	6	13	5	6	8	34
127	5	5	7	9	5	5	6	13	5	5	8	36
255	4	5	7	9	4	5	6	13	4	5	8	37

метър на анизотропия. При $\varepsilon = 0.0$ отново се получава уравнение на Поасон и броят на итерациите е същия като в **Пример 1**. Малко по-големите стойности на N_{it} за $\varepsilon > 0$ се обясняват с увеличаването на вариацията на коефициентите. В случая на най-силно осцилиращи коефициенти за изотропни задачи, именно случая $\varepsilon = 1$ за **Пример 3**, броят на итерациите е най-голям и за двата MP и MV алгоритъма.

За последния пример поведението на N_{it} е подобно, когато $\varepsilon < 1$. В случая $\varepsilon = 1$ първият алгоритъм е неприложим, защото преобусловителят не е положително определен. Вторият алгоритъм работи, но броят на итерациите е 3-4 пъти по-голям от този за $\varepsilon = 0.1$. Това показва, че когато отношението на анизотропия не е известно в явен вид, трябва много внимателно да се прилага MP.

Заключенията от представените резултати и моделния анализ са следните. Предложеният почти оптимален преобусловител е ефективен за големи задачи с не силно изменящи се коефициенти и слаба анизотропия. В случаите, когато едното или и двете условия не са удовлетворени, трябва да се направят някои модификации. Една възможност е да се запазят връзките по направлението на силна анизотропия както е предложено в G. Bencheva, I. Georgiev, S. Margenov [22].

Глава 3

Паралелни преки методи за задачи с разделящи се променливи

Тази глава е посветена на конструиране и изследване на паралелни версии PFASV, PGMF и PGMS на алгоритмите FASV, GMF и GMS, описани в Глава 2. За всяка от тях е отговорено на въпросите: „Как се разделят данни и изчисления?“, „Какви комуникации са необходими?“, „Колко добър е конструираният алгоритъм?“. За PFASV и PGMS са разработени две версии на паралелен SRHS, означени с PSRHSF и PSRHSG, всяка от които е описана при съответния алгоритъм. Получени са теоретични оценки за паралелните времена и ускорение и е направено сравнение на тези оценки за трите алгоритъма върху три от най-често изследваните паралелни архитектури – пръстен, двумерна мрежа и хиперкуб. В края на главата са представени резултати от числените експерименти проведени върху четири паралелни системи – Parmac, Grendel, Beowulf и Blue (вж. секция 1.3) и е сравнено поведението на изследваните алгоритми, както помежду им, така и по отношение на машините.

Резултатите представени в тази глава са публикувани в [19, 20, 21].

3.1 Паралелен бърз алгоритъм за разделяне на променливите

Да предположим, че имаме $N_p = 2^{n_p}$ процесора, т.е. искаме да разделим задачата на N_p задания изпълняващи се едновременно, като $n_p \leq l-1$, $m = 2^l - 1$. Видяхме, че на всяка от стъпките k на правия и обратния ход на Алгоритъм FASV (вж. секция 2.1) се решават непълно различен брой специално конструирани системи. С увеличаване на индекса k броят на тези системи намалява на половина, а размерността им се

увеличава два пъти. Това означава, че в определен момент броят на системите ще стане по-малък от N_p, а блочният им ред – по-голям от $LSTRIP = 2^{l-n_p}$. По такъв начин правият и обратният ход естествено се разделят на два етапа.

Въз основа на тези наблюдения, генерираме **началните данни** по следния начин: Областта се разделя на N_p хоризонтални ивици с приблизително равна дължина, именно $LSTRIP$ за първите $N_p - 1$ и $LSTRIP - 1$ за последната. Те съответстват на хоризонтални блокове от системата $A\,\mathbf{x} = \mathbf{f}$. Всеки процесор съдържа целите матрици T и B (пазят се само ненулевите диагонали) и елементите от дясната страна $\mathbf{f}$ за една ивица.

Прав ход. На първите $l - n_p$ стъпки всички процесори работят, решавайки 2^{l-k-n_p} системи с разредена дясна част чрез SRHS. За да се образува векторът $\mathbf{f}^{(k)}$ за следващата стъпка (вж. (2.1.17)), не повече от една блочна компонента на решението трябва да се изпрати на не повече от един процесор. Именно, всеки процесор $P_i\,, (i \neq 0)$ изпраща първата компонента на решението на P_{i-1}.

На всяка от следващите стъпки $k = l - n_p + 1, l - 1$, блочният ред на $A^{(k,s)}\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)}$ е по-голям от $LSTRIP$, а общият брой на тези системи е по-малък от N_p. Това означава, че ако се използва отново последователен SRHS, част от процесорите ще са без работа или ще се дублират изчисления без да се подобрява производителността. По-точно, на всяка стъпка $k + 1$ ще остават без работа половината от процесорите, работили на стъпка k, а всеки от останалите ще решава голяма система последователно. За да се преодолее този проблем, се реализира паралелно и Алгоритъм SRHS (PSRHSF). Процесорите се разделят на групи, като всяка от тях решава непълно по една система с $A^{(k,s)}$. Всяка група се състои от процесори с последователни индекси. Процесорът с най-малък номер в групата наричаме корен. Искаме в корена да се актуализират ненулевите блокове на дясната страна за следващата стъпка и да се възстановяват необходимите компоненти на решението. За всяка следваща стъпка $k + 1$ се обединяват по две от групите на стъпка k и половината от корените спират да са такива.

Пример: При $l = 4$, $m = 2^4 - 1 = 15$, $N_p = 8$, ненулевите блокове на $\mathbf{f}^{(k)}$, $k = 1, \ldots, 4$ се актуализират от процесорите по следния начин:

	P_0		P_1		P_2		P_3		P_4		P_5		P_6		P_7
1 :	$\mathbf{f}_1^{(1)}$	$\mathbf{f}_2^{(1)}$	$\mathbf{f}_3^{(1)}$	$\mathbf{f}_4^{(1)}$	$\mathbf{f}_5^{(1)}$	$\mathbf{f}_6^{(1)}$	$\mathbf{f}_7^{(1)}$	$\mathbf{f}_8^{(1)}$	$\mathbf{f}_9^{(1)}$	$\mathbf{f}_{10}^{(1)}$	$\mathbf{f}_{11}^{(1)}$	$\mathbf{f}_{12}^{(1)}$	$\mathbf{f}_{13}^{(1)}$	$\mathbf{f}_{14}^{(1)}$	$\mathbf{f}_{15}^{(1)}$
2 :		$\mathbf{f}_2^{(2)}$		$\mathbf{f}_4^{(2)}$		$\mathbf{f}_6^{(2)}$		$\mathbf{f}_8^{(2)}$		$\mathbf{f}_{10}^{(2)}$		$\mathbf{f}_{12}^{(2)}$		$\mathbf{f}_{14}^{(2)}$	
	P_0				P_2				P_4				P_6		
3 :				$\mathbf{f}_4^{(3)}$				$\mathbf{f}_8^{(3)}$				$\mathbf{f}_{12}^{(3)}$			
				P_0								P_4			
4 :								$\mathbf{f}_8^{(4)}$							
								P_0							

С вертикални черти са разделени блоковете по процесори P_i за всяка стъпка. На стъпка 1 са работили всички процесори решавайки по една система чрез SRHS, т.е. всеки от тях можем да разглеждаме като корен. На стъпка 2 процесорите се групират по два решавайки по една система чрез PSRHSF, като единият престава да е корен. На стъпка 3 се обединяват две съседни групи, т.е. 4 процесора с последователни индекси, като броят на корените намалява на половина и т.н. Коренът в новата група трябва да актуализира ненулевите блокове на $\mathbf{f}^{(k)}$ за следващата стъпка.

Само процесорите, които „спират“ да са корени, съдържат частите от $\mathbf{f}^{(k)}$, необходими за пресмятането на ненулевите блокове на $\mathbf{f}^{(k+1)}$. По-точно, всеки от тях съдържа по един ненулев блок за $\mathbf{f}^{(k+1)}$ преди актуализацията му. Така освен компонентата на решението, всеки от тях трябва да изпрати по един блок от $\mathbf{f}^{(k)}$ на процесора, който ще бъде корен за неговата група на стъпка $k+1$. Този път комуникациите са между процесори с номера, различаващи се със степен на двойката. В примера по-горе, когато P_3 „спре“ да е корен, той трябва да изпрати $\mathbf{f}_{2^{l-1}}$ на P_2. Когато P_2 „спре“, той трябва да изпрати променения член $\mathbf{f}_{2^{l-1}}$ на P_0. Правият ход приключва за $l-1$ стъпки.

Обратният ход се изпълнява за l стъпки, отново разделени на два етапа. В първите n_p стъпки, т.е. за $k = l, \ldots, l-n_p+1$, се използва паралелен SRHS, а в останалите – последователен, като на всяка стъпка се възстановяват част от компонентите на решението на изходната задача. На първия етап процесорите от всяка група на стъпка k, решаващи една система чрез PSRHSF, се разделят на две групи за стъпка $k+1$. Броят на корените се увеличава два пъти. Всеки корен възстановява по една компонента на решението на изходната задача и трябва да го изпрати на част от корените за следващите стъпки.

Пример: При $l = 4$, $m = 2^4 - 1 = 15$, $N_p = 8$, на всяка стъпка $k = 4, \ldots, 1$ се търсят $\mathbf{x}_{(2s-1)2^{k-1}}$ за $s = 1, 2, \ldots, 2^{4-k}$. Решението се възстановява на части по следния начин:

4 :								$\mathbf{x}_8^0$							
3 :				$\mathbf{x}_4^0$				$*$				$\mathbf{x}_{12}^4$			
2 :		$\mathbf{x}_2^0$		$*$		$\mathbf{x}_6^2$		$*$		$\mathbf{x}_{10}^4$		$*$		$\mathbf{x}_{14}^6$	
1 :	$\mathbf{x}_1^0$	$*$	$\mathbf{x}_3^1$	$*$	$\mathbf{x}_5^2$	$*$	$\mathbf{x}_7^3$	$*$	$\mathbf{x}_9^4$	$*$	$\mathbf{x}_{11}^5$	$*$	$\mathbf{x}_{13}^6$	$*$	$\mathbf{x}_{15}^7$
	P_0		P_1		P_2		P_3		P_4		P_5		P_8		P_7

Символът $*$ означава, че съответната компонента вече е намерена, горният индекс показва номера на процесора, който я изчислява, а с вертикални черти са разделени данните както са по процесори за съответното k.

На първата стъпка компонентата $\mathbf{x}_{2^{l-1}}$ на решението се изчислява от P_0. Компо-

нентата $\mathbf{x}_{2^{l-1}}$ е необходима, за да се изчислят $\mathbf{x}_{2^{l-2}}$ и $\mathbf{x}_{3.2^{l-2}}$ и тя трябва да се изпрати на P_4. Освен това тя трябва да се изпрати и на P_2 и P_3, защото се използва за пресмятане на десните страни на системите за изчисляване на компонентите $\mathbf{x}_{3.2^{l-3}}$ и $\mathbf{x}_{7.2^{l-4}}$. На следващата стъпка компонентите $\mathbf{x}_{2^{l-2}}$ и $\mathbf{x}_{3.2^{l-2}}$ се намират съответно от корените P_0 и P_4. Те се разпращат по подобен начин, но в групи от по 2^{n_p-1} процесора. Така, освен комуникациите в PSRHSF, е необходимо групи от процесори да получат по един вектор с дължина n. По-точно, във всяка група от 2^{n_p-i} процесора (с последователни индекси) за съответната стъпка i, където $i = 0, \ldots, n_p - 1$, процесорът с най-малък номер src възстановява една компонента на решението. След това коренът я изпраща (т.е. 1 вектор с дължина n) до $n_p - i$ от останалите процесори в групата, чиито номера $dest$ се определят чрез формулите: $dest = src + 2^{n_p-i-1}$ и $dest = src + 2^{n_p-i-1} - 2^j$ за $j = 0, \ldots, n_p - i - 2$. След стъпка $l - n_p + 1$ на обратния ход, всеки процесор съдържа необходимите му данни за намиране на останалата част от решението. Така на стъпки от $l - n_p$ до 1 не са необходими никакви комуникации.

Паралелната версия на FASV описана дотук се записва съкратено както следва:

Алгоритъм PFASV

Стъпка 1. Прав ход:

for $k = 1$ to $l - n_p$

решават се 2^{l-k-n_p} системи с $A^{(k,s)}$ чрез SRHS;

комуникации one_to_one: 1 вектор от ред n за *пресмятане* на $\mathbf{f}^{(k+1)}$;

end {цикъл по k}

for $k = l - n_p + 1$ to $l - 1$

решава се 1 система с $A^{(k,s)}$ чрез PSRHSF;

комуникации one_to_one: 2 вектора от ред n за *пресмятане* на $\mathbf{f}^{(k+1)}$;

end {цикъл по k}

Стъпка 2. Обратен ход:

for $k = l$ downto $l - n_p + 1$ $(i = l - k + 1)$

решава се 1 система с $A^{(k,s)}$ чрез PSRHSF;

комуникации one_to_one: 1 вектор от ред n на $n_p - i$ процесора;

end {цикъл по k}

for $k = l - n_p$ downto 1

решават се 2^{l-k-n_p} системи с $A^{(k,s)}$ чрез SRHS;

НЯМА *комуникации*

end {цикъл по k}

Край {PFASV}.

Паралелен SRHS (PSRHSF). Входните и изходните данни при конструирането на PSRHSF са съобразени с изискванията на PFASV. На всяка от стъпките процесорите се обединяват по $\widetilde{P}$, за да решават паралелно една система чрез PSRHSF. Във всяка група има 1 процесор (корен), съдържащ дясната страна, а всички процесори имат матриците $A^{(k,s)}$. След приключване на работата, коренът трябва да има всички търсени компоненти на решението. **Стъпка 2** на SRHS представлява решаване на $n_k = 2^k - 1$ независими тридиагонални системи и се извършва паралелно. За да може всеки от процесорите в групата да реши $\widetilde{m} = \dfrac{n_k}{\widetilde{P}}$ такива системи, той трябва да има нужните данни. Това са част с дължина $\widetilde{m}$ от всички собствени стойности и от необходимите $\widetilde{d} \leq d + r$ компоненти на всички собствени вектори на матрицата $B_k^{(s)}$ съответстваща на $A^{(k,s)}$. Тъй като всеки процесор съдържа цялата матрица B, той може да намери всичките му необходими собствени стойности и вектори в подготвителния етап за изпълнение на PFASV. Коренът изпраща ненулевите блокове на дясната страна на всеки от останалите процесори в групата. След това всеки процесор P_s изпълнява **Стъпки 1, 2** и **3** на SRHS с $\widetilde{m}$ вместо с m и със своите данни $\mathbf{q}_i^{P_s}, \lambda_i^{P_s}, \beta_i^{P_s}, \eta_i^{P_s}$. По този начин P_s намира части $\mathbf{x}_j^{P_s}$ от необходимите r компоненти на решението $\mathbf{x}_j = \sum_{s=0}^{\widetilde{P}-1} \mathbf{x}_j^{P_s}$ за $j = j_1', j_2', \ldots, j_r'$. Отново се извършват комуникации между процесорите, за да се съберат $\mathbf{x}_j, j = j_1', j_2', \ldots, j_r'$ в корена.

Така предложената паралелна реализация на SRHS записваме накратко във вида:

Алгоритъм PSRHSF

Стъпка 0: *Намиране* на $\{\lambda_i\}_{i=1}^{n_k}$ и на необходимите $\widetilde{d} \leq d+r$ компоненти $\{q_{i,j}\}$, $j \in \{j_1, \ldots, j_d\} \cup \{j_1^{'}, \ldots, j_r^{'}\}$ от всички собствени вектори $\{\mathbf{q}_i\}_{i=1}^{n_k}$ на тридиагоналната матрица $B_k^{(s)}$ (предварителен етап);

Стъпка 1: *Комуникации:*

one_to_all broadcast за $\mathbf{f}_j, j = j_1, \ldots, j_d$;

Стъпка 2: *Изпълняват се* **Стъпки 1, 2** и **3** на SRHS с $\widetilde{m} = \dfrac{n_k}{\widetilde{P}}$ и $\mathbf{q}_i^{P_s}$, $\lambda_i^{P_s}$, $\beta_i^{P_s}, \eta_i^{P_s}$, като се намират $\mathbf{x}_j^{P_s}$, $j = j_1', j_2', \ldots, j_r'$;

Стъпка 3: *Комуникации:*

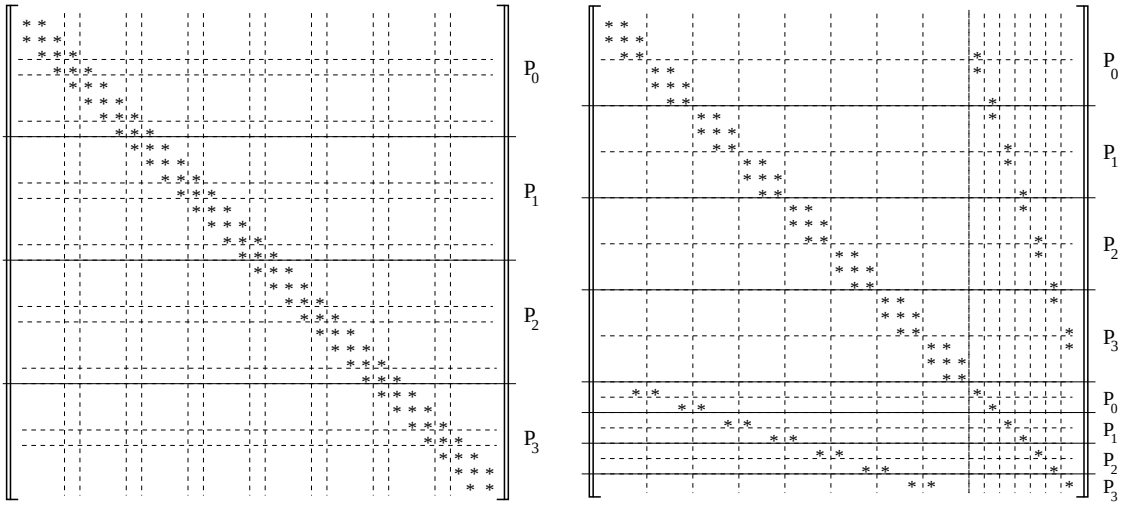
all_to_one reduce за $\mathbf{x}_j^{P_s}$: $\mathbf{x}_j = \sum_{s=0}^{\widetilde{P}-1} \mathbf{x}_j^{P_s}$, $j = j_1', j_2', \ldots, j_r'$

Край {PSRHSF}.

3.2 Паралелни марш алгоритми

Да разгледаме най-напред как GMS (вж. секция 2.2) може да се раздели на малки задания, които се изпълняват едновременно. На всяка от стъпките му има група от независими системи, които могат да се решават паралелно. За **Стъпка 1** и **Стъпка 3** това са p блочни уравнения с матрици $A_s^{(k)}$ от ред kn (k блока от ред n). В **Стъпка 2** тридиагоналните системи $(\lambda_i I_n + T)\eta_i = \beta_i$, $i = 1, \ldots, m$ от ред n могат да се разглеждат като отделни задания. И в двата случая е необходима цялата матрица T. За всяко от заданията в първата група е необходима още и съответната част от матрицата B, докато за втората допълнителните данни са някои от собствените стойности на B и части от собствените ѝ вектори (за да се пресметнат десните страни). Вземайки пред вид горното, разделяме данните и изчисленията по следния начин. Отново предполагаме, че имаме $N_p = 2^{n_p}$ процесора, номерирани с $P_0, P_1, \ldots, P_{N_p-1}$. Допускаме още, че $N_p \leq p$ и няма нужда да се реализира паралелно Алгоритъм SM. Както и при PFASV, разделяме началните данни на N_p ивици с дължина $LSTRIP = 2^{l-n_p}$ за първите $N_p - 1$ и $LSTRIP - 1$ за последната ($m = 2^l - 1 = p(k+1) - 1$). Две ивици с обща граница се съдържат в процесори с последователни индекси. Всеки от P_i, $i = 0, \ldots, N_p - 1$ съдържа целите матрици T и B и i-тия блок на $\mathbf{f}$. Така P_i може да реши $\frac{p}{N_p}$ системи с $A_s^{(k)}$ и да пресметне (на подготвителния етап) необходимите му собствени стойности и вектори. Това разделяне е достатъчно също и за PGMF.

Връзките между уравненията от първоначалната система са в блоковете $\widetilde{A}_{1,2}$ и $\widetilde{A}_{2,1}$ на пренареденатата матрица $\widetilde{A} = \{\widetilde{A}_{i,j}\}_{i,j=1}^{2}$. За да се пресметнат съответните десни части в **Стъпка 1** и **Стъпка 3** трябва да се извършат умножения от вида *матрица* $\times$ *вектор* с $\widetilde{A}_{1,2}$ и $\widetilde{A}_{2,1}$. Как изглеждат тези блокове? На Фигура 3.1 е показана структурата на матриците A и $\widetilde{A}$ и разпределението на елементите им между процесорите в случая $p = 8, k = 3, m = p(k+1), N_p = 4$. С $*$ е означен ненулев блок от ред n. Индексите на елементите в $\widetilde{A}$ са известни и при реализацията на алгоритъма не е необходимо да се пазят блоковете на пренареденатата матрица – затова е представена и изходната матрица A. За $\widetilde{A}$ знаем, че $\widetilde{A}_{1,1}$ и $\widetilde{A}_{2,2}$ са блочно диагонални, като $\widetilde{A}_{1,1}$ има p ненулеви блока от ред kn, а при $\widetilde{A}_{2,2}$ те са $p-1$ на брой и са от ред n. По тази причина, матриците $\widetilde{A}_{1,2}$ и $\widetilde{A}_{2,1}$ имат съответно $p \times (p-1)$ блока от ред $kn \times n$ и $(p-1) \times p$ блока от ред $n \times kn$. В $\widetilde{A}_{1,2}$ всеки стълб има не повече от 2 ненулеви блока от ред n, а при $\widetilde{A}_{2,1}$ това е в сила за редовете. Тогава за пресмятане на $\widetilde{A}_{2,1}\mathbf{y}^{(1)}$, $P_i\,(i \neq 0)$ трябва да изпрати на P_{i-1} първата компонента от ред n от $\mathbf{y}^{(1)}$, а при $\widetilde{A}_{1,2}\mathbf{x}^{(2)}$, $P_i\,(i \neq N_p - 1)$ изпраща на P_{i+1} последния блок от $\mathbf{x}^{(2)}$. Така и в двата случая трябва да се изпрати един вектор с дължина n на процесор със съседен



A $\widetilde{A}$

Фиг. 3.1: Структура на матриците A и $\widetilde{A}$ и разпределение между процесорите в случая $p = 8, k = 3, m = p(k+1), N_p = 4$.

номер.

Разликата между GMS и GMF се състои в начина на решаване на системата с допълнението на Шур, т.е. в **Стъпка 2**. При GMF се използва FASV (част от стъпките му), което означава, че за PGMF е необходим PFASV. При PGMS се използва паралелна реализация PSRHSG на SRHS, при която входните и изходните данни са малко по-различни от тези в PSRHSF и водят до различни комуникации.

Вече сме готови да представим схематично алгоритмите PGMF и PGMS:

Алгоритъм PGM

Стъпка 1: *решаване* на $\dfrac{p}{N_p}$ системи с $A_s^{(k)}$ чрез Алгоритъм SM;
комуникации one_to_one: 1 вектор от ред n;
пресмятане на $\widetilde{\mathbf{f}}^{(2)}$;

Стъпка 2: *Непълно решаване* на $A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}$ (вж. (2.2.28)) чрез:
Алгоритъм PSRHSG (за PGMS) или
Алгоритъм PFASV (за PGMF);

Стъпка 3: *комуникации* one_to_one: 1 вектор от ред n;
пресмятане на $\widetilde{\mathbf{f}}^{(1)}$;
решаване на $\dfrac{p}{N_p}$ системи с $A_s^{(k)}$ чрез Алгоритъм SM;

Край {PGM}.

Алгоритъм PFASV беше описан в предишната секция. Сега ще се спрем по-подробно на Алгоритъм PSRHSG. Както казахме, тук входните и изходните данни са разпределени по друг начин в сравнение с PSRHSF. В началото на **Стъпка 2** на PGMS всеки процесор има блок от дясната страна и целите матрици T и B. На края P_i съдържа група от $\frac{p}{N_p}$ от търсените компоненти $\mathbf{x}_{s(k+1)}$, $s = 1, \ldots, p-1$ (за P_{N_p-1} са $\frac{p}{N_p} - 1$). Това води до друга организация на комуникациите.

Алгоритъм PSRHSG

Стъпка 0: *Намиране* на $\lambda = \{\lambda_i\}_{i=1}^m$ и на необходимите $\widetilde{d} \leq d + r$ компоненти $\{q_{i,j}\}$, $j \in \{j_1, \ldots, j_d\} \cup \{j_1^{'}, \ldots, j_r^{'}\}$ от всички собствени вектори $\{\mathbf{q}_i\}_{i=1}^m$ на тридиагоналната матрица B (предварителен етап);

Стъпка 1: *Намиране* на коефициентите на Фурие $\beta_{i,j}$ чрез:

Изчисления: $\beta_{i,j}^{(Pr)} = \sum_{s=1}^{d(Pr)} q_{j_s,j} f_{i,j_s}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$;

Комуникации: all_to_all reduce_scatter за β_j;

Стъпка 2: *решаване* на $(\lambda_j I_n + T)\, \eta_j = \beta_j$, $j = 1, \ldots, \frac{m}{N_p}$;

Стъпка 3: *възстановяване* на $\mathbf{x}_j$, $j = j_1', j_2', \ldots, j_r'$ чрез:

Изчисления: $\mathbf{x}_j^{(Pr)} = \sum_{i=1}^{\widetilde{m}} q_{j,i}\eta_i$, $j = j_1', \ldots, j_r', \widetilde{m} = \frac{m}{N_p}$;

Комуникации: all_to_all reduce_scatter за $\mathbf{x}_j$

Край {PSRHSG}.

И тук предполагаме, че на подготвителния етап всеки от процесорите е намерил необходимите му компоненти от собствените стойности и вектори. За **Стъпка 2** всеки процесор трябва да има по $\frac{m}{N_p}$ от собствените стойности, т.е. блок с дължина $\frac{m}{N_p}$ от вектора $\lambda = \{\lambda_i\}_{i=1}^m$. За **Стъпка 1**, т.е. за да пресметне коефициентите $\beta_{i,j}$, всеки процесор се нуждае от $d(Pr) = \frac{d}{N_p}$ компоненти от всички собствени вектори. За **Стъпка 3** на всеки процесор са необходими по r компоненти от $\frac{m}{N_p}$ собствени вектори. Изчисленията и комуникациите в **Стъпка 1** са същите, както ако трябва да се извърши умножение на матрици, разпределени по следния начин:

$$\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right) \left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right) = \left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right).$$

$$F_{n\times d} \qquad Q_{d\times m} \qquad Beta_{n\times m}$$

Тук лявата матрица е разделена на ивици по стълбове, дясната – по редове и резултатната матрица трябва да е разпределена отново на ивици по стълбове. Всеки процесор пресмята частите $\beta_{i,j}^{(Pr)}$ от $\beta_{i,j}$ и след това в $P_i, i = 0, \ldots, N_p - 1$ се събират по $\frac{m}{N_p}$ от $\beta_j = \sum_{Pr} \beta_j^{(Pr)}$. Необходимите комуникации, означени с all_to_all reduce_scatter, се извършват по следния начин – първо се изпълнява all_to_one reduce и един процесор съдържа всички β_j, а след това чрез one_to_all scatter се изпращат съответните групи на процесорите, които ще ги използват. За следващата **Стъпка 2** всеки процесор трябва да реши $\frac{m}{N_p}$ независими тридиагонални системи без никакви комуникации. Третата стъпка се извършва подобно на първата с единствената разлика, че втората матрица и резултатът сега са вектор-стълбове

$$\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right) \left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right) = \left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right).$$

$$Q_{r\times m} \qquad V_m \qquad X_r$$

Отново се намират части, този път от $\mathbf{x}_j$ и след това с all_to_all reduce_scatter всеки процесор получава необходимите му компоненти $\mathbf{x}_j = \sum_{Pr} \mathbf{x}_j^{(Pr)}$. Всички комуникации в PSRHSG (както и в PSRHSF) са глобални.

3.3 Оценки на паралелните времена и ускорения

В точки 3.1 и 3.2 бяха предложени три паралелни алгоритъма за решаване на задача (1.1.6) с разделящи се променливи. Тук са получени оценки за времената за изчисления и комуникации за всеки от тях в общ вид и като илюстрация е сравнено поведението им върху три архитектури на паралелни изчислителни системи – пръстен, квадратна мрежа и хиперкуб. В края на секцията са обобщени характеристиките на предложените по-горе методи.

Нека имаме паралелна система с $N_p = 2^{n_p}$ процесора и нека са в сила предположенията направени в т. 1.3.2. Нека освен това $m+1 = p(k+1)$, $p = 2^{l_p}$, $k+1 = 2^{l_k}$, $l = l_p + l_k$ (l, l_p, $l_k \in \mathbb{Z}$). Допускаме също, че процесори със съседни номера са съседи и физически, т.е. комуникациите между тях са локални.

Ще се спрем най-напред на **PSRHSF** и **PSRHSG**, тъй като се различават малко един от друг и всеки от алгоритмите използва една от двете паралелни реализации на Алгоритъм SRHS. Нека търсим r компоненти на решението на задача от ред mn с разредена дясна част с d ненулеви блока, като използваме N_p процесора. За краткост, с $T_a^{psf}(N_p, m, r, d)$ и $T_a^{psg}(N_p, m, r, d)$ означаваме времената за изчисление, а с $T_{com}^{psf}(N_p, m, r, d)$ и $T_{com}^{psg}(N_p, m, r, d)$ – времената за комуникации, съответно за PSRHSF и PSRHSG.

Изчисления: В PSRHSF (секция 3.1) се извършват **Стъпки 1, 2** и **3** на SRHS с $\frac{m}{N_p}$ вместо с m. Като използваме **Теорема 2.1.2** получаваме, че $T_a^{psf}(N_p, m, r, d) = \frac{m}{N_p}(2(r+d)n + 5n - 4) * t_a = \frac{\mathcal{N}_{SRHS}}{N_p} * t_a$. При PSRHSG (секция 3.2) в **Стъпка 1** се извършват $2\frac{d}{N_p}mn$ ар. оп. за пресмятане на съответните коефициенти на Фурие. В **Стъпка 2** всеки процесор решава $\frac{m}{N_p}$ системи линейни уравнения като извършва $\frac{m}{N_p}(5n-4)$ ар. оп. Решението се намира в **Стъпка 3** чрез $2r\frac{m}{N_p}n$ ар. оп. Общо времето за изчисление на PSRHSG е $T_a^{psg}(N_p, m, r, d) = \left(2\frac{mn}{N_p}(r+d) + \frac{m}{N_p}(5n-4)\right) * t_a = \frac{\mathcal{N}_{SRHS}}{N_p} * t_a$. С други думи, в двата паралелни варианта на SRHS времето за изчисления е едно и също и е разделено по равно между процесорите.

Какво е положението с **комуникациите**? При PSRHSF чрез one_to_all broadcast се изпращат ненулевите компоненти на дясната страна, т.е. d вектора с дължина n за време $b(N_p, d.n)$ (вж. т. 1.3.2). За да се намери решението, трябва да се извърши и all_to_one reduce за r вектора с дължина n, т.е. необходимо е време $b(N_p, r.n)$. При PSRHSG се извършват две операции all_to_all reduce_scatter. В **Стъпка 1** чрез нея се изпращат части от коефициентите на Фурие – първо се извършва all_to_one reduce за m вектора, всеки с дължина n за време $b(N_p, m.n)$ и след това one_to_all scatter на пакети съдържащи $\frac{m}{N_p}$ вектора с дължина n за време $s(N_p, \frac{m}{N_p}n)$. В **Стъпка 3** по подобен начин се прехвърлят части от решението за време $b(N_p, r.n) + s(N_p, \frac{r}{N_p}n)$. Така получаваме, че е в сила следното твърдение.

Теорема 3.3.1 *Двете паралелни реализации PSRHSF и PSRHSG на* Алгоритъм SRHS *за решаване на задача от вида (1.1.6) с разредена дясна част с d ненулеви компоненти като се търсят r компоненти на решението използвайки N_p процесора се изпълняват съответно за времена $T^{psf}(N_p,\, m,\, r,\, d)$ и $T^{psg}(N_p,\, m,\, r,\, d)$, където*

$$
\begin{aligned}
T^{psf}(N_p,\, m,\, r,\, d) &= T_a^{psf}(N_p,\, m,\, r,\, d) + T_{com}^{psf}(N_p,\, m,\, r,\, d)\,,\\
T^{psg}(N_p,\, m,\, r,\, d) &= T_a^{psg}(N_p,\, m,\, r,\, d) + T_{com}^{psg}(N_p,\, m,\, r,\, d)\,,\\
T_a^{psf}(N_p,\, m,\, r,\, d) &= T_a^{psg}(N_p,\, m,\, r,\, d) = \left(2\frac{mn}{N_p}(r+d) + \frac{m}{N_p}(5n-4)\right) * t_a\,,\\
T_{com}^{psf}(N_p,\, m,\, r,\, d) &= b(N_p,\, d.n) + b(N_p,\, r.n)\,,\\
T_{com}^{psg}(N_p,\, m,\, r,\, d) &= b(N_p,\, m.n) + s(N_p,\, \frac{m}{N_p}n) + b(N_p,\, r.n) + s(N_p,\, \frac{r}{N_p}n)\,.
\end{aligned}
$$

С помощта на изразите за времената $b(N_p,\, M)$ и $s(N_p,\, M)$ представени в Таблица 1.1 получаваме, че за пръстен, квадратна мрежа и хиперкуб (означени съответно с горни индекси r, m, h) са в сила оценките:

$$
\begin{aligned}
T_{com}^{psf,\,r}(N_p,\, m,\, r,\, d) &= 2\left\lceil\frac{N_p}{2}\right\rceil * t_s + \left\lceil\frac{N_p}{2}\right\rceil c_1 * t_w\,,\\
T_{com}^{psg,\,r}(N_p,\, m,\, r,\, d) &= 2\Big(N_p - 1 + \left\lceil\frac{N_p}{2}\right\rceil\Big) * t_s + \Big(c_2 + \left\lceil\frac{N_p}{2}\right\rceil c_3\Big) * t_w\,,\\
T_{com}^{psf,\,m}(N_p,\, m,\, r,\, d) &= 4\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil * t_s + 2\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil c_1 * t_w,\\
T_{com}^{psg,\,m}(N_p,\, m,\, r,\, d) &= 4\Big(\sqrt{N_p} - 1 + \left\lceil\frac{\sqrt{N_p}}{2}\right\rceil\Big) * t_s + \Big(c_2 + 2\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil c_3\Big) * t_w,\\
T_{com}^{psf,\,h}(N_p,\, m,\, r,\, d) &= 2\log N_p * t_s + c_1 \log N_p * t_w\,,\\
T_{com}^{psg,\,h}(N_p,\, m,\, r,\, d) &= 4\log N_p * t_s + (c_2 + c_3 \log N_p) * t_w\,,
\end{aligned}
$$

където $c_1 = (d+r)n$, $c_2 = \dfrac{N_p - 1}{N_p}(m+r)n$, $c_3 = (m+r)n$.

Бърз алгоритъм за разделяне на променливите. Като се използват изразите за $T_a^{psf}(N_p,\, m,\, r,\, d)$ и $T_{com}^{psf}(N_p,\, m,\, r,\, d)$, за изчисленията и комуникациите на Алгоритъм PFASV получаваме ($n_i = 2^i - 1$):

Прав ход – изчисления: На стъпки $i = 1, \ldots, l - n_p$ всеки от процесорите решава 2^{l-i-n_p} системи с блочен ред $2^i - 1$ и пресмята съответните десни страни. Аналогично на **Теорема 2.1.3** получаваме, че необходимите за целта ар. оп. са

$$
\mathcal{N}_f^1 = \sum_{i=1}^{l-n_p} 2^{l-n_p-i}\big((13n-4)(2^i-1) + 4n\big) \sim (13n-4)(l-n_p)2^{l-n_p} - (2^{l-n_p}-1)(9n-4)\,.
$$

За следващите стъпки $i = l - n_p + 1, \ldots, l - 1$, т.е. за $j = 1, \ldots, n_p - 1$ процесорите се разделят на $2^{n_p - j}$ групи от по 2^j процесора. Всяка група решава една система от ред $\widehat{m} = \dfrac{m+1}{2^{n_p-j}} - 1$ като използва PSRHSF и пресмята съответните десни страни. Необходимите ар. оп. в този случай са:

$$\mathcal{N}_f^2 = \sum_{j=1}^{n_p-1} \left(T_a^{psf}(2^j,\, \widehat{m},\, 3,\, 1) + 4n\right) \sim (n_p - 1)\Big((13n-4)\frac{m}{N_p} + 4n\Big)\,.$$

Общо изчислителната сложност на правия ход на PFASV е

$$\mathcal{N}_f^{pfasv} = \mathcal{N}_f^1 + \mathcal{N}_f^2 \sim \frac{m}{N_p}\big((l-1)(13n-4) - 9n\big)\,,$$

а времето за изчисления е $T_{a,f}^{pfasv}(N_p,\, m) = \mathcal{N}_f^{pfasv} * t_a$.

В алгоритъм PGMF се извършват $l_p = l - l_k$ стъпки $i = l_k + 1, \ldots, l - 1$ от правия ход на PFASV и изчислителната сложност в този случай е:

$$\begin{aligned}
\mathcal{N}_{f,inc}^1 &= \sum_{i=l_k+1}^{l-n_p} 2^{l-n_p-i}\big((13n-4)(2^i-1) + 4n\big) \\
&\sim (13n-4)(l-n_p-l_k)2^{l-n_p} - (2^{l-n_p-l_k}-1)(9n-4)\,, \\
\mathcal{N}_{f,inc}^2 &= \sum_{j=1}^{n_p-1} \left(T_a^{psf}(2^j,\, \widehat{m},\, 3,\, 1) + 4n\right) \sim (n_p-1)\Big((13n-4)\frac{m}{N_p} + 4n\Big)\,, \\
\mathcal{N}_{f,inc}^{pfasv} &= \mathcal{N}_{f,inc}^1 + \mathcal{N}_{f,inc}^2 \sim \frac{m}{N_p}(l-l_k-1)(13n-4) - 9\frac{p}{N_p}n\,.
\end{aligned}$$

За времето имаме $T_{a,\,f,\,inc}^{pfasv}(N_p,\, m) = \mathcal{N}_{f,inc}^{pfasv} * t_a$.

Прав ход – комуникации: За стъпки $i = 1, \ldots, l - n_p$ всеки от P_s, $s \neq 0$ изпраща на P_{s-1} един вектор с дължина n и времето за комуникации се оценява както следва:

$$T_{com}^{f,1} \leq \sum_{i=1}^{l-n_p}(t_s + n * t_w) = (l-n_p)(t_s + n * t_w)\,.$$

Съответната част за PGMF изисква

$$T_{com,inc}^{f,1} \leq \sum_{i=l_k+1}^{l-n_p}(t_s + n * t_w) = (l-n_p-l_k)(t_s + n * t_w)\,.$$

За стъпки $i = l - n_p + 1, \ldots, l - 1$, т.е. за $j = 1, \ldots, n_p - 1$, освен комуникациите в PSRHSF, всеки процесор трябва да изпрати 2 вектора с дължина n (по една компонента на решението и на дясната страна) на несъседен процесор. В този случай времето за комуникации се оценява с

$$T_{com}^{f,2} \leq \sum_{j=1}^{n_p-1} \left(T_{com}^{psf}(2^j,\, \widehat{m},\, 3,\, 1) + 2\,oo(2^j,\, n)\right)\,.$$

Общо необходимото време за комуникации на правия ход е

$$\begin{aligned} T^{pfasv}_{com,\,f}(N_p,\,m) &= T^{f,1}_{com} + T^{f,2}_{com} \\ &\leq (l-n_p)(t_s + n * t_w) + \sum_{j=1}^{n_p-1} \big(T^{psf}_{com}(2^j,\,\widehat{m},\,3,\,1) + 2\,oo(2^j,\,n)\big), \end{aligned}$$

$$\begin{aligned} T^{pfasv,inc}_{com,\,f}(N_p,\,m) &= T^{f,1}_{com,\,inc} + T^{f,2}_{com} \\ &\leq (l_p-n_p)(t_s + n * t_w) + \sum_{j=1}^{n_p-1} \big(T^{psf}_{com}(2^j,\,\widehat{m},\,3,\,1) + 2\,oo(2^j,\,n)\big). \end{aligned}$$

Обратен ход – изчисления: Аналогично на правия ход, за стъпки $i = l, \ldots, l - n_p + 1$ и $i = l - n_p, \ldots, 1$ са необходими съответно $\mathcal{N}^1_b$ и $\mathcal{N}^2_b$ ар. оп., където

$$\begin{aligned} \mathcal{N}^1_b &= \sum_{j=1}^{n_p-1} \big(T^{psf}_a(2^j,\,\widehat{m},\,1,\,2) + 2n\big) + T^{psf}_a(N_p,\,m,\,1,\,1) + 2n \\ &\sim \frac{m}{N_p}\big((n_p-1)(11n-4) + (9n-4)\big)\,, \\ \mathcal{N}^2_b &= \sum_{i=1}^{l-n_p} 2^{l-n_p-i}\big((11n-4)(2^i-1) + 2n\big) \\ &\sim (11n-4)(l-n_p)2^{l-n_p} - (2^{l-n_p}-1)(9n-4)\,, \end{aligned}$$

Общо изчислителната сложност на обратния ход на PFASV е

$$\mathcal{N}^{pfasv}_b = \mathcal{N}^1_b + \mathcal{N}^2_b \sim \frac{m}{N_p}(l-1)(11n-4)\,,$$

а времето за изчисления е $T^{pfasv}_{a,b}(N_p,\,m) = \mathcal{N}^{pfasv}_b * t_a$.

За стъпките изпълнявани при PGMF са необходими $\mathcal{N}^1_{b,inc} = \mathcal{N}^1_b$ и

$$\begin{aligned} \mathcal{N}^2_{b,inc} &= \sum_{i=l_k+1}^{l-n_p} 2^{l-n_p-i}\big((11n-4)(2^i-1) + 2n\big) \\ &\sim (11n-4)(l-n_p-l_k)2^{l-n_p} - (2^{l-n_p-l_k}-1)(9n-4) \end{aligned}$$

ар. оп. или общо

$$\mathcal{N}^{pfasv}_{b,inc} = \mathcal{N}^1_{b,inc} + \mathcal{N}^2_{b,inc} \sim \frac{m}{N_p}(l-l_k-1)(11n-4) + 9n\frac{m-p}{N_p}\,,$$

откъдето времето за изчисление е $T^{pfasv}_{a,b,inc}(N_p,\,m) = \mathcal{N}^{pfasv}_{b,inc} * t_a$.

Обратен ход – комуникации: На стъпки $i = l, \ldots, l - n_p + 1$, т.е. за $j = 1, \ldots, n_p$ един от процесорите във всяка група от 2^j процесора изпраща до j от останалите

(не са му съседи), един вектор с дължина n. Освен това се извършват и съответните комуникации от PSRHSF. На останалите стъпки няма комуникации. Следователно $T^{pfasv}_{com,b}(N_p,\, m) = T^{pfasv,inc}_{com,b}(N_p,\, m) = T^{b,1}_{com}$ и

$$T^{b,1}_{com} \le \sum_{j=1}^{n_p-1} \left(T^{psf}_{com}(2^j,\, \widehat{m},\, 1,\, 2) + j\, oo(2^j,\, n)\right) + T^{psf}_{com}(2^{n_p},\, m,\, 1,\, 1) + n_p\, oo(2^{n_p},\, n)\,.$$

Обобщаваме тези резултати в следната теорема.

Теорема 3.3.2 *Паралелната реализация на* FASV, *т.е.* Алгоритъм PFASV, *се изпълнява за време* $T^{pfasv}(N_p,\, m) = T^{pfasv}_{a}(N_p,\, m) + T^{pfasv}_{com}(N_p,\, m)$, *където*

$$\begin{aligned}
T^{pfasv}_{a}(N_p,\, m) &= T^{pfasv}_{a,f}(N_p,\, m) + T^{pfasv}_{a,b}(N_p,\, m) = \left(24n\frac{m}{N_p}(l-1) - 9n\frac{m}{N_p}\right) * t_a,\\
T^{pfasv}_{com}(N_p,\, m) &= T^{pfasv}_{com,f}(N_p,\, m) + T^{pfasv}_{com,b}(N_p,\, m)\\
&\le (l-n_p)(t_s + n * t_w) + T^{psf}_{com}(2^{n_p},\, m,\, 1,\, 1) + n_p\, oo(2^{n_p},\, n)\\
&+ \sum_{j=1}^{n_p-1} \left(T^{psf}_{com}(2^j,\, \widehat{m},\, 3,\, 1) + T^{psf}_{com}(2^j,\, \widehat{m},\, 1,\, 2) + (2+j)\, oo(2^j,\, n)\right).
\end{aligned}$$

Виждаме, че и тук аритметичните операции са разделени по равно между процесорите. За разглежданите архитектури, пръстен, квадратна мрежа и хиперкуб, са в сила следните оценки за времето за комуникации:

$$\begin{aligned}
T^{pfasv,r}_{com}(N_p,\, m) &\le \left(l - 6 + 3.2^{n_p} + \frac{n_p(n_p+3)}{2}\right) * t_s\\
&+ n\left(l - n_p - 8 + 2^{n_p-1}(2n_p+9)\right) * t_w\,,\\
T^{pfasv,m}_{com}(N_p,\, m) &\le \left(l - 6 + \lfloor f_1(n_p)\rfloor + \frac{n_p(n_p+19)}{2}\right) * t_s\\
&+ n\left(l + 13n_p - 10 + \lfloor f_2(n_p)\rfloor\right) * t_w\,,\\
T^{pfasv,h}_{com}(N_p,\, m) &\le \left(l - 2 + \frac{n_p(5n_p+3)}{2}\right) * t_s + n\left(l + \frac{n_p(n_p^2+15n_p-10)}{3}\right) * t_w\,,
\end{aligned}$$

където $f_1(n_p) = -4(2+\sqrt{2}) + 2\sqrt{2}^{n_p}(3+2\sqrt{2})$ и $f_2(n_p) = -14 - 6\sqrt{2} + \sqrt{2}^{n_p}(7 + 6\sqrt{2} + (2+\sqrt{2})n_p)$.

Обобщен марш алгоритъм. Двата варианта PGMF и PGMS, а следователно и времената им за изчисления и комуникации, се различават по **Стъпка 2**. В PGMF участват съответно $T^{pfasv}_{a,inc}(N_p, m)$ и $T^{pfasv}_{com,inc}(N_p, m)$, а в PGMS това са $T^{psg}_{a}(N_p, m, p-1, p-1)$ и $T^{psg}_{com}(N_p, m, p-1, p-1)$. Какви са изчисленията и комуникациите на **Стъпки 1** и **3**? При всяка от тях се решават $\dfrac{p}{N_p}$ системи от ред k чрез Алгоритъм SM, т. е. се извършват $\dfrac{p}{N_p}31kn$ ар. оп., а за пресмятане на дясната страна са необходими още $4n\dfrac{(p-1)}{N_p}$ ар. оп. Общо за **Стъпки 1** и **3** изчисленията са $\mathcal{N}^{pgm}_{1\&3} \approx 62\dfrac{pk}{N_p}n$.

За да се пресметнат десните страни, на всяка от тези стъпки се изпраща по 1 вектор с дължина n на съседен процесор, т.е. времето за комуникации за **Стъпки 1** и **3** е $T^{pgm}_{com,1\&3} = 2(t_s + n * t_w)$. Общо времето за изпълнение на **Стъпки 1** и **3** е $T^{pgm}_{1\&3}(N_p,\, m) = 62\frac{pk}{N_p}n*t_a+2(t_s+n*t_w)$. Така за PGMF и PGMS имаме $T^{pgmf}(N_p,\, m) = T^{pgm}_{1\&3}(N_p,\, m)+T^{pfasv}_{inc}(N_p,\, m)$ и $T^{pgms}(N_p,\, m) = T^{pgm}_{1\&3}(N_p,\, m)+T^{psg}(N_p,\, m, p-1, p-1)$. Като се използват получените оценки за съответните изрази, се доказва следното твърдение.

Теорема 3.3.3 *Паралелната реализация PGMF на обобщения марш алгоритъм в комбинация с бързия алгоритъм за разделяне на променливите се изпълнява за време* $T^{pgmf}(N_p,\, m) = T^{pgmf}_a(N_p,\, m) + T^{pgmf}_{com}(N_p,\, m)$. *Модификацията PGMS, използваща само техниката за непълно решаване на задачи с разредена дясна част, изисква време* $T^{pgms}(N_p,\, m) = T^{pgms}_a(N_p,\, m) + T^{pgms}_{com}(N_p,\, m)$. *За събираемите в тези изрази са в сила релациите:*

$$
\begin{aligned}
T^{pgmf}_a(N_p,\, m) &= \Big(62\frac{pk}{N_p}n + 24n\frac{m}{N_p}(l_p-1) + 9n\frac{m-2p}{N_p}\Big) * t_a\,,\\
T^{pgms}_a(N_p,\, m) &= \Big(62\frac{pk}{N_p}n + \frac{m}{N_p}n(4p+1)\Big) * t_a\,,\\
T^{pgmf}_{com}(N_p,\, m) &\le (l_p - n_p + 2)(t_s + n * t_w) + T^{psf}_{com}(2^{n_p},\, m,\, 1,\, 1) + n_p\, oo(2^{n_p},\, n)\\
&+ \sum_{j=1}^{n_p-1}\big(T^{psf}_{com}(2^j,\, \widehat{m},\, 3,\, 1) + T^{psf}_{com}(2^j,\, \widehat{m},\, 1,\, 2) + (2+j)\, oo(2^j,\, n)\big),\\
T^{pgms}_{com}(N_p,\, m) &\le 2(t_s + n * t_w) + T^{psg}_{com}(N_p,\, m, p-1, p-1)\,.
\end{aligned}
$$

Отново изчисленията са разделени по равно между процесорите. За пръстен, квадратна мрежа и хиперкуб, времената за комуникации за PGMF се оценяват с

$$
\begin{aligned}
T^{pgmf,r}_{com}(N_p,\, m) &\le \Big(l_p - 4 + 3.2^{n_p} + \frac{n_p(n_p+3)}{2}\Big) * t_s\\
&+ n\big(l_p - n_p - 6 + 2^{n_p-1}(2n_p+9)\big) * t_w\,,\\
T^{pgmf,m}_{com}(N_p,\, m) &\le \Big(l_p - 4 + \lfloor f_1(n_p)\rfloor + \frac{n_p(n_p+19)}{2}\Big) * t_s\\
&+ n\big(l_p + 13n_p - 8 + \lfloor f_2(n_p)\rfloor\big) * t_w\,,\\
T^{pgmf,h}_{com}(N_p,\, m) &\le \big(l_p + \frac{n_p(5n_p+3)}{2}\big) * t_s + n\big(l_p + 2 + \frac{n_p(n_p^2+15n_p-10)}{3}\big) * t_w,
\end{aligned}
$$

като $f_1(n_p)$ и $f_2(n_p)$ са дефинирани по-горе. За PGMS тези времена са

$$T_{com}^{pgms,r}(N_p,\, m) \leq 2\Big(N_p + \left\lceil \frac{N_p}{2} \right\rceil\Big) * t_s + \Big(2n + g\big(c + \left\lceil \frac{N_p}{2} \right\rceil\big)\Big) * t_w\,,$$

$$T_{com}^{pgms,m}(N_p,\, m) \leq \Big(4\sqrt{N_p} - 2 + 4\left\lceil \frac{\sqrt{N_p}}{2} \right\rceil\Big) * t_s + \Big(2n + g\big(c + 2\left\lceil \frac{\sqrt{N_p}}{2} \right\rceil\big)\Big) * t_w,$$

$$T_{com}^{pgms,h}(N_p,\, m) \leq (4\log N_p + 2) * t_s + \big(2n + g(c + \log N_p)\big) * t_w\,,$$

където $g = (m + p - 1)n$ и $c = \dfrac{N_p - 1}{N_p}$.

Кой от предложените паралелни алгоритми е най-добър? Времето за изпълнение на всеки от тях, а следователно ускорението $S_{N_p} = \dfrac{T_1}{T_{N_p}}$ и ефективността $E_{N_p} = \dfrac{S_{N_p}}{N_p}$, зависи както от пресмятанията, така и от количеството и типа на необходимите комуникации.

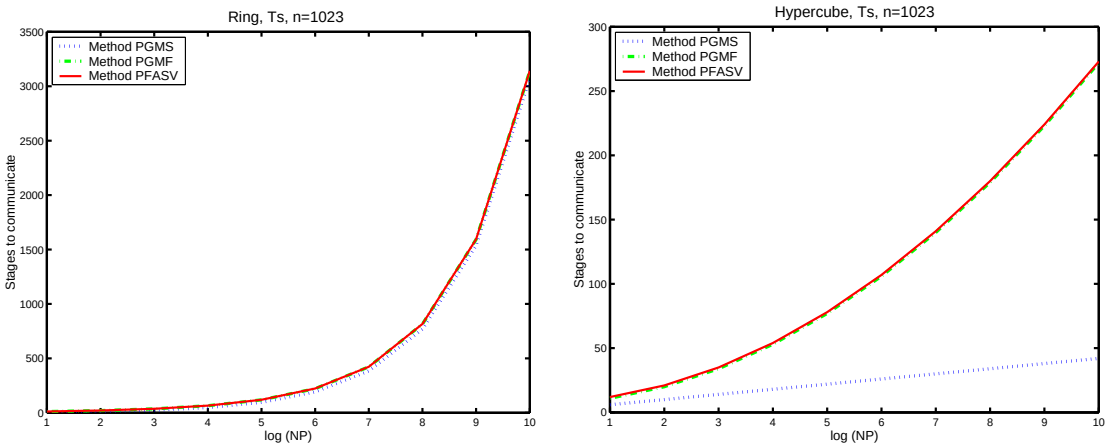
При системи с обща памет, доминиращо е времето за изчисления $T_a(N_p)$ ($T^{Alg}(N_p) = T_a(N_p) + T_{com}(N_p)$). И при трите изследвани алгоритъма, аритметичните операции се разпределят по равно между процесорите и $T_a(2\,N_p) = \dfrac{1}{2}\,T_a(N_p)$. При слабо влияние на $T_{com}(N_p)$, алгоритъмът с най-малка изчислителна сложност, т.е. PGMF, е за предпочитане.

При системи с разпределена памет, производителността на алгоритмите зависи от $T_{com}(N_p)$. За всеки от разглежданите методи, получихме оценки от вида $T_{com}(N_p) \leq T_s * t_s + T_w * t_w$ за три архитектури – пръстен, квадратна мрежа и хиперкуб. Коефициентите T_s и T_w отразяват съответно броя на етапите, при които са необходими комуникации и общия брой прехвърлени думи. Те зависят допълнително и от архитектурата, тъй като при разглежданите алгоритми имаме както локални, така и глобални комуникации.

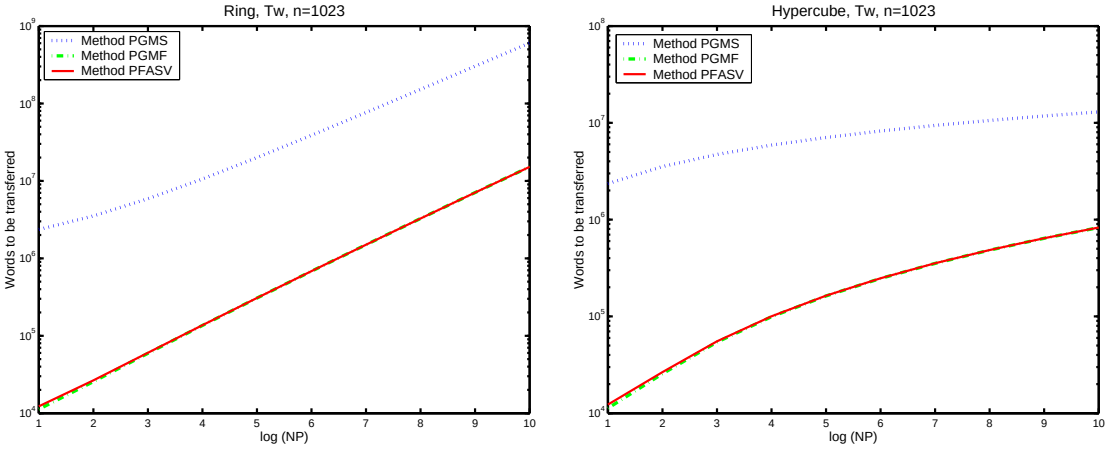
На Фигура 3.2 е изобразен T_s като функция на n_p ($N_p = 2^{n_p}$) за $m = n = 1023$ за пръстен и хиперкуб. Графиките за трите алгоритъма при пръстен почти съвпадат, като PGMS има слабо предимство. Поведението на T_s при мрежа е подобно, тъй като водещите членове (както и при пръстен) за трите алгоритъма са равни. При хиперкуб, както се вижда от Фигура 3.2, T_s^{pgms} расте много по-бавно отколкото за останалите алгоритми. Причината е, че тук водещият член е n_p, а при PGMF и PFASV той е n_p^2.

За всички случаи T_w зависи както от броя на процесорите, така и от размерността на задачата. На Фигура 3.3 е показано поведението на T_w за $m = n = 1023$ – графиката на T_w^{pgms} винаги е над другите две.

Основното предимство на PGMS е, че етапите, на които се изискват комуникации



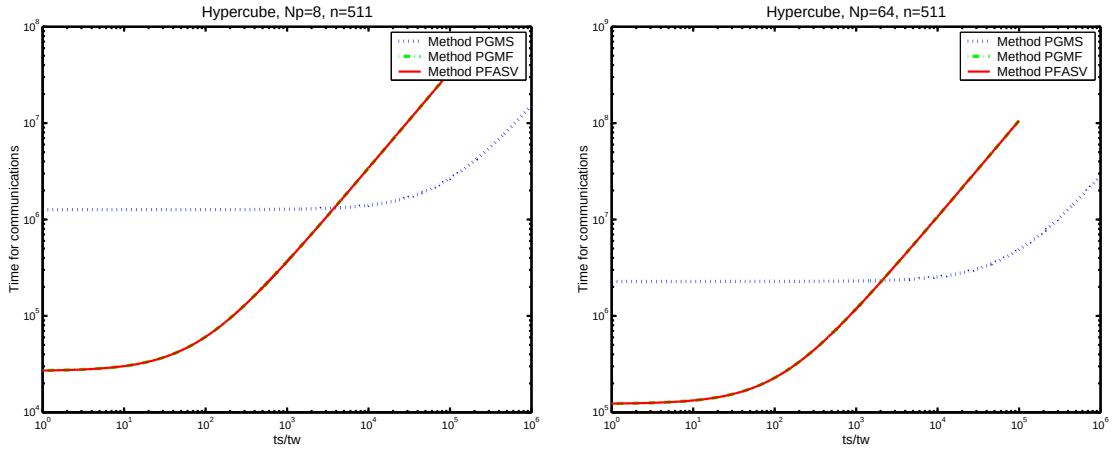
Фиг. 3.2: Коефициенти пред t_s, за $k = 7$, $n = 1023$.



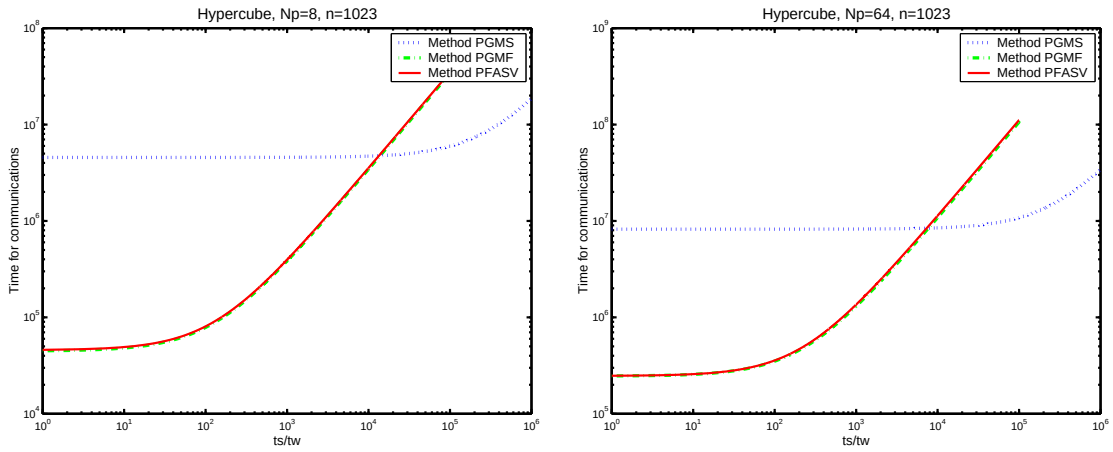
Фиг. 3.3: Коефициенти пред t_w, за $k = 7$, $n = 1023$.

са константа (общо 4) и не зависят от изпълнението на програмата (т.е. са статични). При PFASV (както и при PGMF) на част от стъпките (кои точно зависи от размерността на задачата и броя на процесорите) се извършват „глобални“ за група процесори комуникации. Големината на тази група и количеството данни отново зависят от mn и N_p и се определят динамично, което отнема допълнително време и води до намаляване на ефективността.

Върху общото време за комуникации влияят и параметрите t_s и t_w на конкретната паралелна система ($t_s \gg t_w > t_a$). На Фигури 3.4 и 3.5 е изобразено T_{com} за хиперкуб като функция на $\tau = \dfrac{t_s}{t_w} \in [1,\ 10^6)$ за $n = 511,\ 1023$ и $N_p = 8,\ 64$. За малки стойности на τ, T_{com}^{pfasv} и T_{com}^{pgmf} ($\approx T_{com}^{pfasv}$) растат бавно и са много по-малки от T_{com}^{pgms}. При $\tau \approx 10^2$, T_{com}^{pgmf} започва бързо да нараства и в определен момент τ_0 пресича графиката на T_{com}^{pgms}. Колко е τ_0 зависи и от броя на процесорите, и от размерността на задачата. За представените случаи имаме $10^3 \leq \tau_0 \leq 10^4$. Имайки пред вид всичко казано до тук,



Фиг. 3.4: Времена за комуникации като функция на $\frac{t_s}{t_w}$ за $k = 7$, $n = 511$.



Фиг. 3.5: Времена за комуникации като функция на $\frac{t_s}{t_w}$ за $k = 7$, $n = 1023$.

можем да очакваме, че за системи с малки стойности на t_w, t_s и t_s/t_w алгоритмите PFASV и PGMF ще имат по-добра производителност. Асимптотично относно броя на процесорите, PGMS може да е по-добър за някои паралелни системи (напр. клъстери, мрежи от работни станции), както по отношение на реалното време за изпълнение, така и спрямо ускорението.

3.4 Числени експерименти

В тази секция се представят резултатите от използването на паралелните алгоритми PFASV, PGMF и PGMS за решаване на задача (1.1.6) върху Grendel, Beowulf, Parmac и Blue (вж. т. 1.3.3). За Parmac са разгледани и двете възможности за изпълнение на MPI програма – с опция N и без нея, означени съответно с Parmac(N) и Parmac.

Всички представени времена в тази секция са в секунди и са най-добрите получени при 3 изпълнения на програмата върху дадена машина.

За да сравним поведението на изследваните в тази глава паралелни алгоритми, използваме следния частен случай на задача (1.1.2):

Пример: Коефициентите са $a_1(y_1) = 1 + y_1^2$, $a_2(y_2) = e^{-y_2}$. За решение $u(y_1, y_2)$ се избира функцията $u(y_1, y_2) = (1 - y_1)y_1y_2(1 - y_2)$ и следователно дясната страна е $f(y_1, y_2) = 2y_2(1-y_2)(3y_1^2 - y_1 + 1) + e^{-y_2}y_1(1-y_1)(3-2y_2)$. Предполагат се хомогенни гранични условия на Дирихле, т.е. $\mu(y) \equiv 0$.

Съответната дискретна задача има $N = n^2$ степени на свобода. Тя се получава чрез диференчна апроксимация по петточков шаблон „кръст“ върху равномерна $n \times n$ мрежа (т.е. $m = n$) със стъпка $h = \dfrac{1}{n+1}$. Използваме разлагането $n + 1 = p(k + 1)$ за различни p и $k \in \mathbb{Z}$. Всички алгоритми са директни и грешката на решението е всъщност грешката от диференчната апроксимация плюс грешка от закръгляне (вж. секция 2.3).

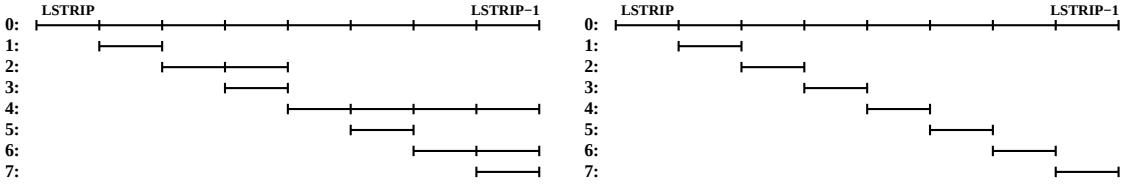
При написването на преносими MPI програми за паралелни системи от различен тип (обща памет, разпределена памет, клъстери от системи с обща памет) е важно да се отчете следната особеност. В MPI е заложено (с подходящи настройки при инсталирането), че комуникациите се извършват съобразно това дали системата е с обща или с разпределена памет. Когато се дублират изчисления върху система с обща памет, MPI дублира и съответните данни.

По тази причина при програмната реализация на паралелните алгоритми предложени и изследвани в тази глава са направени следните модификации:

1) Собствените стойности и вектори се намират в момента, в който потрябват, а не се пазят през цялото изпълнение на програмите;

2) При алгоритмите PSRHSF и PSRHSG само коренът съдържа цялата матрица $A^{(k,s)}$, а останалите процесори в групата имат по ивица с дължина $LSTRIP$ от нея. Коренът намира компонентите на собствените стойности и вектори и с помощта на една глобална комуникация one_to_all scatter ги изпраща на останалите процесори в групата за време $s(N_p, (r + d + 1)\dfrac{m}{N_p})$.

3) Всеки от процесорите за PFASV, PGMF и PGMS съдържа една или няколко ивици от B (а не цялата матрица), с цел да може да намери необходимите му собствени стойности и вектори. Дублирането на данни е илюстрирано на Фигура 3.6. При PGMS е необходимо P_0 да съдържа цялата матрица B, а за останалите процесори е достатъчна по една ивица от B. При PFASV и PGMF дублирането на данни е съобразено с начина на нарастване на блочния ред на $A^{(k,s)}$. Например, нека имаме



а) PFASV, PGMF б) PGMS

Фиг. 3.6: Начални данни във всеки процесор, $N_p = 8$.

$N_p = 2^3$ процесора, номерирани с P_0, P_1, $\ldots P_{N_p-1} \equiv P_7$. Всеки нечетен процесор съдържа елементите на матрицата B съответстващи на нечетните ивици. При четните процесори данните се дублират – P_0 съдържа цялата матрица B, P_2 и P_6 съдържат данните от две последователни ивици, а P_4 съдържа втората половина на B. Разпределението за 16 процесора ще получим като отчетем, че P_0 съдържа всички ивици и „копираме“ два пъти структурата от Фиг. 3.6 а) – веднъж за P_0 до P_7 и втори път за P_8 до P_{15}. В общия случай, всеки четен процесор съдържа α ивици, като α се определя така: P_0 съдържа N_p ивици, P_{2j} съдържа α ивици, ако е в сила $\alpha = (2j) \bmod 2^{\alpha+1}$ за $j = 2^{n_p-2}, 2^{n_p-3}, \ldots, 2^0$.

Тези модификации не влияят на оценките получени в т. 3.3, тъй като времената за намиране и разпределяне на собствени стойности и компоненти на собствени вектори не са включени в общите времена за изпълнение на алгоритмите. В същото време, те позволяват да се използват ефективно и клъстери от системи с обща памет.

Най-напред ще се спрем на поведението на алгоритмите върху Parmac, Grendel и Beowulf. В Таблица 3.1 са сравнени времената за изчисление на последова-

Таблица 3.1: Резултати за последователните алгоритми.

Platform	n	FASV	GMF		GMS	
			k=3	k=7	k=3	k=7
Grendel	255	3.12	3.02	2.90	1.99	1.02
	511	19.01	19.29	11.66	18.64	7.94
	1023	87.62	87.49	73.61	117.79	86.03
Beowulf	255	6.91	6.95	4.02	4.48	2.53
	511	28.57	28.42	16.88	23.04	12.79
	1023	121.33	120.67	74.45	133.99	72.65
Parmac	255	3.57	3.54	2.07	2.44	1.23
	511	15.60	15.42	9.20	13.81	6.39
	1023	65.55	65.29	39.94	86.94	40.20

телните алгоритми. В първата колона е дадена машината, а във втората – стойността на n. Третата колона показва процесорното време за алгоритъм FASV. Следващите четири колони са групирани по две и дават времената за GMF и GMS за две различни стойности на k, $k = 3$ и $k = 7$. За по-голямо k стандартният марш алгоритъм SM за този пример е неустойчив. Как устойчивостта на SM влияе върху грешката на GMF и GMS е показано в секция 2.3.

Виждаме, че времената на Grendel като цяло са по-малки от тези на Beowulf, но са по-големи от тези на Parmac (с изключение на $n = 255$). И за трите машини, поведението на изследваните алгоритми си прилича. Времената за $k = 7$, както за GMF, така и за GMS са по-добри от тези за $k = 3$. За задачи с малка размерност ($n = 255, n = 511$), Алгоритъм GMS е най-бърз. Както се очакваше, за $n = 1023$ той става по-бавен. Времената за FASV са приблизително равни на тези за $k = 3$ на GMF, защото в този случай GMF изпълнява почти всички стъпки на FASV.

В следващите четири таблици са представени измерените паралелни времена и получените ускорение и ефективност върху Grendel, Beowulf, Parmac и Parmac(N). Форматът им е един и същ: първата колона характеризира размерността на задачата – дадено е n, където броят на неизвестните е $N = n^2$. Втората колона показва броя N_p на използваните процесори. Останалите колони са в групи по три – по една за всеки алгоритъм. Във всяка група първата колона е за измереното cpu-време. За вариантите на паралелния GM алгоритъм е представен случаят $k = 7$. Втората колона във всяка група е за ускорението $S_{N_p} = \dfrac{T_2}{T_{N_p}}$, а третата – за ефективността $E_{N_p} = \dfrac{2\,T_2}{N_p\,T_{N_p}}$. Теоретичните горни граници за ускорението и ефективността са $S_{N_p} \leq \dfrac{N_p}{2}$ и $E_{N_p} \leq 1$.

Да видим най-напред какво се случва на Grendel (Таблица 3.2). Тенденцията за всеки от алгоритмите е, че S_{N_p} и E_{N_p} растат при увеличаване на размерността на задачата и $E_4 > E_8$. Наблюдава се допълнително отрицателно влияние върху ускорението (и ефективността) за $N_p = 8$ в случая $n = 1023$, защото Grendel има само 8 процесора. Алгоритъмът с най-добро ускорение за $N_p = 4$ се сменя при промяна на стойността на n. За $N_p = 8$ „победителят“ винаги е PGMS. Времената за изпълнение на PGMS са най-малки за всички стойности на n и N_p с изключение на $n = 1023$, $N_p = 2$.

Да видим сега и резултатите от Beowulf (Таблица 3.3). Отново cpu-времената на PGMS са най-малки за всички случаи, но поведението на ускорението и ефективността е коренно различно. За алгоритмите PFASV и PGMF ефективността намалява с увеличаване на n, докато за PGMS тя се увеличава. „Победителят“ отново се променя, но PGMS е губещ за $n = 255, 511$ и $N_p = 4, 8$. За големи задачи, т.е. $n = 1023$,

PGMS има най-добра ефективност и за двете стойности на $N_p = 4, 8$.

Таблица 3.2: Резултати за паралелните алгоритми върху Grendel, $k = 7$.

		PFASV			PGMF			PGMS		
n	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	2	2.47	–	–	1.60	–	–	1.07	–	–
255	4	1.42	1.74	0.87	0.89	1.80	0.90	0.63	1.70	0.85
	8	1.08	2.29	0.57	0.65	2.46	0.62	0.37	2.89	0.72
	2	16.86	–	–	10.52	–	–	7.59	–	–
511	4	8.85	1.91	0.96	5.73	1.84	0.92	3.95	1.92	0.96
	8	4.98	3.39	0.85	3.28	3.21	0.80	2.24	3.39	0.85
	2	79.59	–	–	49.53	–	–	50.96	–	–
1023	4	41.63	1.91	0.96	26.13	1.90	0.95	24.00	2.12	1.06
	8	29.38	2.71	0.68	16.19	3.06	0.77	12.49	4.08	1.02

Таблица 3.3: Резултати за паралелните алгоритми върху Beowulf, $k = 7$.

		PFASV			PGMF			PGMS		
n	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	2	8.94	–	–	6.31	–	–	2.29	–	–
255	4	3.90	2.29	1.15	2.72	2.32	1.16	1.74	1.32	0.66
	8	2.31	3.87	0.97	1.55	4.07	1.02	1.11	2.06	0.52
	2	47.52	–	–	36.02	–	–	13.28	–	–
511	4	23.40	2.03	1.02	17.83	2.02	1.01	8.51	1.56	0.78
	8	15.44	3.07	0.77	13.35	2.70	0.67	8.14	1.63	0.41
	2	271.13	–	–	227.04	–	–	89.46	–	–
1023	4	165.57	1.64	0.82	148.74	1.53	0.77	50.45	1.77	0.89
	8	126.88	2.14	0.53	125.18	1.81	0.45	41.07	2.18	0.55

При Parmac (Таблица 3.4) отново с нарастване на размерността на задачата се увеличават ускорението и ефективността. Тук PGMS е с най-малка ефективност, независимо от това, че времето за изпълнение за $n = 255, 511$ е най-добро. Алгоритъмът с най-добро ускорение за $N_p = 8$ е PFASV, а за $N_p = 4$ се мени при различни стойности на n.

При Parmac(N) (Таблица 3.5) поведението на алгоритмите е почти същото. Разликата е, че тук PFASV е с най-добро ускорение за $n = 255, 511$ и при двете стойности $N_p = 4$ и $N_p = 8$, докато за $n = 1023$ „победител“ е PGMF. Като цяло, времената за

изпълнение са по-малки от тези на Parmac. За $N_p = 8$ те са почти равни, въпреки очакванията, че времето за комуникации ще е по-голямо. Причината е в по-бавния достъп до паметта (вж. E. I. Atanassov [10]) – когато два съседни процеса са в един възел, те използват обща памет за комуникациите. В същото време, те се конкурират за достъп до данните и това оказва по-голямо неблагоприятно влияние върху общото време за изпълнение.

Таблица 3.4: Резултати за паралелните алгоритми върху Parmac, $k = 7$.

		PFASV			PGMF			PGMS		
n	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	2	2.36	–	–	1.34	–	–	0.68	–	–
255	4	1.30	1.82	0.91	0.74	1.81	0.91	0.54	1.26	0.63
	8	0.73	3.23	0.81	0.51	2.63	0.66	0.54	1.26	0.32
	2	10.12	–	–	5.82	–	–	4.02	–	–
511	4	5.34	1.90	0.95	3.11	1.87	0.94	2.54	1.58	0.79
	8	3.11	3.25	0.81	1.80	3.23	0.81	2.25	1.79	0.45
	2	42.14	–	–	25.16	–	–	25.44	–	–
1023	4	22.88	1.84	0.92	13.33	1.89	0.95	15.08	1.69	0.85
	8	11.96	3.52	0.88	7.54	3.34	0.84	12.06	2.11	0.53

Таблица 3.5: Резултати за паралелните алгоритми върху Parmac(N), $k = 7$.

		PFASV			PGMF			PGMS		
n	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	2	1.86	–	–	1.06	–	–	0.62	–	–
255	4	1.00	1.86	0.93	0.60	1.77	0.89	0.57	1.09	0.55
	8	0.73	2.55	0.64	0.50	2.12	0.53	0.55	1.13	0.28
	2	7.97	–	–	4.70	–	–	3.52	–	–
511	4	4.43	1.80	0.90	2.61	1.80	0.90	2.56	1.38	0.69
	8	2.95	2.70	0.68	1.96	2.40	0.60	2.32	1.52	0.38
	2	33.57	–	–	22.14	–	–	23.16	–	–
1023	4	17.45	1.92	0.96	10.76	2.06	1.03	14.74	1.57	0.79
	8	11.86	2.83	0.71	7.21	3.07	0.77	11.51	2.01	0.50

В следващите три Таблици 3.6, 3.7 и 3.8 са сравнени времената за PGMF и PGMS на Grendel, Parmac и Parmac(N) за $n = 511\,(l = 9)$ и $n = 1023\,(l = 10)$. Поради технически проблеми с Beowulf нямахме възможност да попълним подобна таблица

Таблица 3.6: Сравнение на PGMF и PGMS върху Grendel, $k = 7$.

		PGMF					PGMS				
l	N_p	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c
	2	10.52	4.10	6.42	0.04	0.11	7.59	4.31	3.28	0.02	0.11
9	4	5.73	2.15	3.58	0.05	0.29	3.95	2.16	1.79	0.05	0.20
	8	3.28	1.16	2.12	0.10	0.40	2.24	1.11	1.13	0.05	0.30
	2	50.85	19.85	31.00	0.89	0.51	50.96	19.24	31.72	0.08	0.40
10	4	26.69	10.22	16.47	0.91	0.73	24.00	9.31	14.69	0.11	0.77
	8	16.19	6.50	9.69	1.93	1.42	12.49	4.76	7.73	0.11	1.34

Таблица 3.7: Сравнение на PGMF и PGMS върху Parmac, $k = 7$.

		PGMF					PGMS				
l	N_p	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c
	2	5.82	2.46	3.36	0.01	0.02	4.02	2.45	1.57	0.01	0.14
9	4	3.11	1.22	1.89	0.01	0.15	2.54	1.21	1.33	0.01	0.95
	8	1.80	0.61	1.19	0.01	0.07	2.25	0.65	1.60	0.05	1.47
	2	25.16	10.26	14.90	0.02	0.01	25.44	10.27	15.17	0.02	0.45
10	4	13.33	5.08	8.25	0.02	0.28	15.08	5.09	9.99	0.01	3.73
	8	7.54	2.52	5.02	0.01	0.68	12.06	2.68	9.38	0.19	6.83

Таблица 3.8: Сравнение на PGMF и PGMS върху Parmac(N), $k = 7$.

		PGMF					PGMS				
l	N_p	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c	T_{N_p}	T_r	T_{s2}	T_r^c	T_{s2}^c
	2	4.70	1.92	2.78	0.01	0.02	3.52	1.91	1.61	0.01	0.43
9	4	2.61	1.07	1.54	0.13	0.04	2.56	0.96	1.60	0.02	1.27
	8	1.96	0.61	1.35	0.01	0.23	2.32	0.63	1.69	0.04	1.54
	2	22.14	8.16	13.98	0.02	0.82	23.16	8.16	15.00	0.03	1.85
10	4	10.76	4.00	6.76	0.02	0.40	14.74	4.00	10.74	0.02	5.31
	8	7.21	2.49	4.72	0.07	0.42	11.51	2.62	8.89	0.15	6.54

и за него. Втората колона показва броя на процесорите. Останалите колони образуват две групи – по една за всеки от алгоритмите. Времето за целия алгоритъм е представено в първата колона, а следващите две във всяка група показват съответно времето T_r за **Стъпка 1** и **Стъпка 3** и T_{s2} за **Стъпка 2**. Последните две колони са за времената за комуникации за **Стъпка 1** и **3** (T_r^c) и за **Стъпка 2** (T_{s2}^c). Времето T_r е почти едно и също за двете версии на GM и в трите таблици. На Grendel и Parmac, за по-малката задача **Стъпка 2** отнема по-малко време, докато на Parmac(N) тя е по-бавна. Времето за комуникации на Grendel не е много голямо и не влияе на ускорението толкова, колкото общият брой на процесорите му. За Parmac и Parmac(N) комуникациите отнемат по-голямата част от времето на **Стъпка 2** за PGMS. Разликата между използването на обща и разпределена памет се вижда във времената за комуникации на Parmac и Parmac(N) за $N_p = 2$. Времето за изчисления за **Стъпка 2** ($T_{s2} - T_{s2}^c$) за PGMS (и за трите таблици) намалява приблизително на половина при увеличаване на процесорите два пъти, докато при PGMF не е така. Това показва, колко важно значение за производителността имат компилатора, организацията и скоростта на достъпа до паметта, както и времето необходимо за извикване на функции в програмния код (вж. също т. 2.3.3).

В последните две таблици са показани резултати от Blue – представител на друг клас паралелни компютри с повече от 1000 процесора. В Таблица 3.9 са дадени вре-

Таблица 3.9: Резултати за паралелните алгоритми върху Blue, $n = 1023$.

	PFASV		PGMF				PGMS			
			$k=3$		$k=7$		$k=3$		$k=7$	
N_p	T_{N_p}	E_{N_p}	T_{N_p}	E_{N_p}	T_{N_p}	E_{N_p}	T_{N_p}	E_{N_p}	T_{N_p}	E_{N_p}
4	131.36	–	111.92	–	64.77	–	70.20	–	40.12	–
8	97.93	0.67	72.24	0.78	40.93	0.79	34.15	1.03	19.99	1.00
16	80.79	0.41	51.34	0.55	29.47	0.55	17.84	0.98	12.27	0.82
32	79.61	0.21	43.51	0.32	21.70	0.37	8.53	1.03	5.73	0.88
64	57.73	0.14	27.06	0.26	10.63	0.38	4.86	0.90	3.23	0.78
128	46.46	0.09	14.79	0.24	12.35	0.16	4.26	0.52	3.33	0.38

мената T_{N_p} и ефективността $E_{N_p} = \dfrac{4\,T_4}{N_P\,T_{N_p}}$ за трите алгоритъма, за $k = 3$ и $k = 7$. Форматът ѝ е подобен на първите две таблици в секцията. Тук най-добро поведение има PGMS за $k = 3$.

На компютри като Blue са интересни и друг тип експерименти. Дискретната задача се разделя на части с равна големина и всеки процесор съдържа една от тях.

Таблица 3.10: Резултати върху Blue – фиксирана ивица $m_l = 16$, $m + 1 = N_p * m_l$.

			PFASV	PGMF		PGMS	
n	m	N_p		$k = 3$	$k = 7$	$k = 3$	$k = 7$
	63	4	2.96	2.90	1.60	1.36	0.73
	127	8	3.59	3.33	2.02	1.42	0.78
511	255	16	4.73	3.82	2.51	1.62	0.92
	511	32	16.89	8.04	3.45	2.02	1.18
	1023	64	53.01	13.21	5.25	2.27	1.65
	2047	128	119.22	36.21	21.56	3.91	2.64
	63	4	7.39	6.24	3.26	2.73	1.49
	127	8	10.90	8.01	4.07	2.87	1.61
1023	255	16	17.48	10.79	5.53	3.22	1.92
	511	32	31.57	15.91	6.91	3.83	2.36
	1023	64	59.91	27.40	9.08	4.43	4.10
	2047	128	216.12	48.40	15.66	7.38	5.16

С увеличаване на броя на процесорите расте размерността на задачата. Времето за изпълнение трябва да остава едно и също, ако имаме максимално ефективен паралелен алгоритъм. В Таблица 3.10 са представени резултати от такъв експеримент. Задачата е разделена на ивици по m с дължина $m_l = 16$, т.е. $m + 1 = N_p * m_l$, като сега имаме $m \neq n$ и n е фиксирано. При PGMS времето се променя най-бавно и като цяло той се изпълнява най-бързо.

Изводът направен в предишната секция, че понякога най-бавният от разглежданите последователни алгоритми може да има най-добро паралелно поведение се потвърди от получените (и представени тук) експериментални резултати.

Паралелните свойства на FASV са изследвани също от Sv. Petrova [75] и от T. Rossi, J. Toivanen [80], но не са ни известни работи отнасящи се за паралелни марш алгоритми. Задачата, разгледана от Sv. Petrova [75] е уравнение на Поасон, дискретизирано върху равномерна мрежа. Изчислителната област също е разделена на ивици, но разпаралелването на FASV в тази работа е направено при предположение, че двойките собствени стойности и вектори са предварително известни в явен вид. По тази причина не е необходимо да се решава задача за намирането им и всеки процесор съдържа данните за една ивица. Ние разглеждаме по-общ случай и нашите резултати не могат пряко да се сравняват с тези представени в Sv. Petrova [75].

В T. Rossi, J. Toivanen [80], граничната задача на Дирихле за уравнението на

Поасон се дискретизира върху неравномерна мрежа и всеки процесор съдържа десните страни от една ивица от разделянето на областта. Решението на възникващите задачи за собствени стойности и вектори е предварителна стъпка. По такъв начин е необходима допълнителна памет за съхранение на всички собствени стойности и нужните компоненти от всички собствени вектори вместо за началните данни, както е в нашия подход. Тестовете в T. Rossi, J. Toivanen [80] са изпълнени на паралелен компютър Cray T3E-750 с разпределена памет и 64 Alpha 21164 (EV5) 375 MHz процесора като е използван MPI стандарт. За 8 процесора са получени ефективности 0.87, 0.94, 0.95, 0.98, когато n е съответно 255, 511, 1023 и 2047. С 64 процесора е постигната ефективност около 0.70 за $n = 1023$.

Глава 4

Паралелни итерационни методи

Настоящата последна част от дисертацията е посветена на теоретично и експериментално изследване на нов паралелен преобусловител. С негова помощ се решават ефективно големи СЛАУ от вида (1.1.10), получени след дискретизация на изходната двумерна елиптична гранична задача от втори ред (1.1.1) чрез завъртени билинейни неконформни крайни елементи върху четириъгълници. В основата на предложения алгоритъм от тип спрегнат градиент с преобуславяне е модифицираната непълна факторизация на Холецки (Modified Incomplete Cholesky, MIC(0)) на разредени матрици. Тя е приложена върху локално конструирана апроксимация на изходната матрица на коравина, позволяваща устойчива MIC(0) факторизация. Полученият преобусловител има специална блочна структура, която запазва изчислителната ефективност на поточковата непълна факторизация като едновременно с това дава възможност за паралелна реализация. Свойствата на паралелния алгоритъм са илюстрирани с числени експерименти върху два Beowulf клъстера. Тяхната архитектура показва потенциала на разработения паралелен алгоритъм за реализации върху системи с обща памет.

Резултатите представени в тази глава са публикувани в [24, 25]. Те са приложени от G. Bencheva, I. Georgiev, S. Margenov [22] при конструирането на паралелен двунивов преобусловител за анизотропни задачи, дискретизирани чрез Crouzeix-Raviart триъгълни неконформни елементи. По-точно, допълнението на Шур за предложеното в G. Bencheva, I. Georgiev, S. Margenov [22] блочно представяне на съответната матрица на коравина има подобна структура и паралелните свойства на двата алгоритъма си приличат.

4.1 Стратегия на преобуславянето

В тази секция е представена същността на MIC(0) и на предложената поелементна конструкция на преобусловителя. Повече информация относно MIC(0) може да се намери в [26, 51, 52].

Реалната симетрична матрица A може да се представи като сума на диагонална (D), строго долно- $(-\widetilde{L})$ и горно- $(-\widetilde{L})^t$ триъгълни части

$$A = D - \widetilde{L} - \widetilde{L}^t \,. \tag{4.1.1}$$

Да разгледаме следната приближена факторизация на A

$$\mathcal{C}_{\mathrm{MIC}(0)}(A) = (X - \widetilde{L})X^{-1}(X - \widetilde{L})^t \,, \tag{4.1.2}$$

където $X = diag(x_1, \ldots, x_N)$ е диагонална матрица определена от условието за равни суми на елементите по редове

$$\mathcal{C}_{\mathrm{MIC}(0)}(A)\mathbf{e} = A\mathbf{e} \,, \quad \mathbf{e} = (1, \ldots, 1)^t \in \mathbf{R}^N \,. \tag{4.1.3}$$

Казва се, че (4.1.2), (4.1.3) е *устойчива* MIC(0) факторизация на A, ако $X > 0$ и следователно $\mathcal{C}_{\mathrm{MIC}(0)}(A)$ е положително определена. Относно конструкцията и устойчивостта на MIC(0) факторизацията е в сила следната теорема.

Теорема 4.1.1 *Нека $A = \{a_{ij}\}$ е симетрична реална $N \times N$ матрица и $A = D - \widetilde{L} - \widetilde{L}^t$. Освен това, нека (неравенствата са покомпонентни)*

$$\begin{aligned} \widetilde{L} &\geq 0 \,, \\ A\mathbf{e} &\geq 0 \,, \\ A\mathbf{e} + \widetilde{L}^t\mathbf{e} &> 0 \,, \quad \mathbf{e} = (1, \cdots, 1)^t \in \mathbf{R}^N \,, \end{aligned}$$

т. е. A е слабо диагонално доминираща матрица с неположителни извъндиагонални елементи и $A + \widetilde{L}^t = D - \widetilde{L}$ е строго диагонално доминираща. Тогава

$$x_i = a_{ii} - \sum_{k=1}^{i-1} \frac{a_{ik}}{x_k} \sum_{j=k+1}^{N} a_{kj}$$

са положителни и матрицата $X = diag(x_1, \cdots, x_N)$ дефинира устойчива MIC(0) *факторизация на A.*

Забележка 4.1.1 *Числените експерименти представени в тази глава са изпълнени използвайки пертурбирана версия на* MIC(0) *алгоритъма, където непълната*

факторизация е приложена върху матрицата $\widetilde{A} = A + \widetilde{D}$. *Диагоналната пертурбация* $\widetilde{D} = \widetilde{D}(\xi) = diag(\widetilde{d}_1, \ldots \widetilde{d}_N)$ *е дефинирана по следния начин:*

$$\widetilde{d}_i = \begin{cases} \xi a_{ii} & \text{при} \quad a_{ii} \geq 2w_i \\ \xi^{1/2} a_{ii} & \text{при} \quad a_{ii} < 2w_i \end{cases} ,$$

където

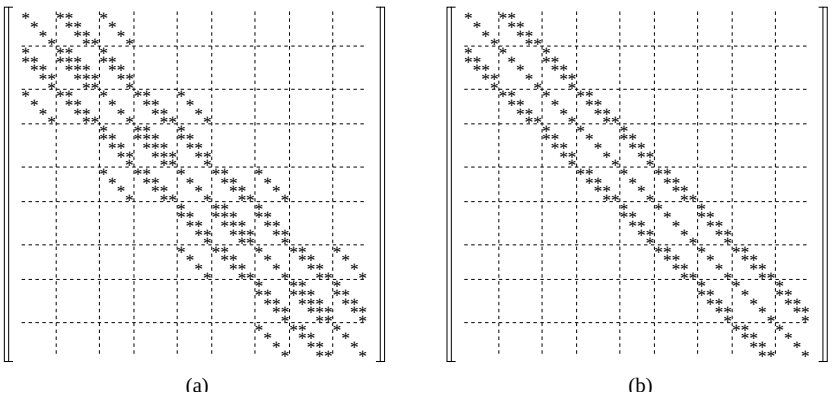
$$w_i = -\sum_{j>i} a_{ij} .$$

Тук $0 < \xi < 1$ *е константа от порядъка на минималната собствена стойност на* A. *Изчисленията за разгледаните моделни задачи са с* $\xi = h^2$.

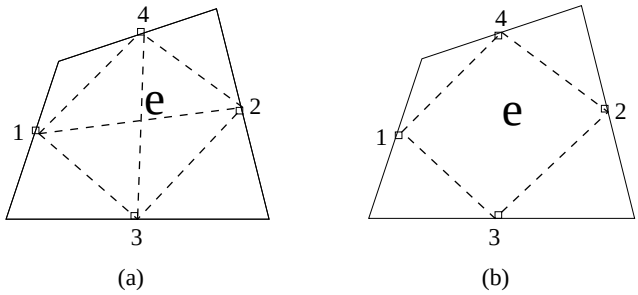
За да се реши системата с преобусловителя $\mathcal{C}_{\mathrm{MIC}(0)}(A)$, трябва да се реши една система с долно-триъгълна матрица, една – с горно-триъгълна и една – с диагонална матрица. Две от тези стъпки се основават на рекурсивни изчисления и по тази причина резултатния PCG алгоритъм в основата си е последователен. За да може да се направи паралелна реализация, се въвежда локално конструирана апроксимация B на изходната матрица на коравина A. Предложеният подход е в сила и за двата варианта (MP и MV) възлови базисни функции, въведени в т. 1.1 в зависимост от интерполационния оператор – поточков или интегрален. За да се илюстрират основните идеи на алгоритъма, се разглежда задача в квадратна област Ω. Използва се стандартно ориентирана правоъгълна мрежа, като номерирането на неизвестните следва колоните от възлите. Нека n_i е броят на неконформните елементи по направление y_i. Тогава размерността на матрицата на коравина A след налагане на граничните условия е $N = n_1(2n_2 + 1) + n_2$. Следвайки стандартната процедура за асемблиране в МКЕ, матрицата A се записва във вида

$$A = \sum_{e \in \omega_h} L_e^T A_e L_e .$$

Тук A_e е елементната матрица на коравина, а L_e е рестрикцията на глобалния вектор на неизвестните върху текущия четириъгълен елемент e. Структурата на матрицата A, съответстваща на такава моделна задача е показана на Фигура 4.1(а). Вижда се, че A има не повече от седем ненулеви елемента на ред и само пет от тях са с постоянни индекси по отношение на кодиагоналите. Останалите сменят алтернативно положението си, променяйки ненулевата структура на блочните редове на B. Редуват се пет ненулеви блока и диагонален блок по главния диагонал с три ненулеви блока и тридиагонален блок по главния диагонал. Схемата на връзките съответстващи на матрицата на коравина A_e за даден завъртян билинеен неконформен краен елемент



Фиг. 4.1: Структура на (a) матрицата A и (b) въведената матрица B.



Фиг. 4.2: Схема на връзките на (a) матрицата A_e и (b) матрицата B_e.

е представена на Фигура 4.2(a). Диагоналите на пунктираните четириъгълници се изобразяват в ненулевите елементи с „променлива позиция“. Така ако се „отрежат“ тези връзки в A_e, съответните елементи в глобалната матрица на коравина ще се анулират. Следвайки тази идея, се въвежда локално модифицирана елементна матрица на коравина B_e със схема на връзките показана на Фигура 4.2(b). Компонентите на B_e се дефинират така:

$$A_e = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}, \quad B_e = \begin{bmatrix} \beta_{11} & 0 & \alpha_{13} & \alpha_{14} \\ 0 & \beta_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \beta_{33} & 0 \\ \alpha_{41} & \alpha_{42} & 0 & \beta_{44} \end{bmatrix}, \tag{4.1.4}$$

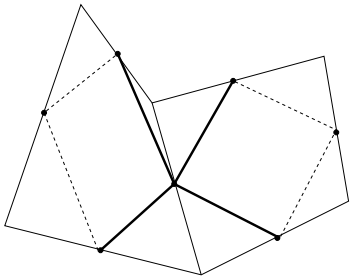
където

$$\beta_{11} = \alpha_{11} + \alpha_{12}\,,\ \beta_{22} = \alpha_{22} + \alpha_{21}\,,\ \beta_{33} = \alpha_{33} + \alpha_{34}\,,\ \beta_{44} = \alpha_{44} + \alpha_{43}\,,$$

т. е. A_e и B_e имат равни суми на елементите по редове. Това означава, че $Ker(A_e) = Ker(B_e)$. Чрез поелементно асемблиране на дефинираните матрици B_e, се получава глобална матрица

$$B = \sum_{e \in \omega_h} L_e^T B_e L_e$$

с регулярна блочна структура илюстрирана на Фигура 4.1(b). Структурата на B може да се интерпретира като наклонен петточков шаблон (вж. Фигура 4.3), като в



Фиг. 4.3: Наклонен петточков шаблон.

най-обща постановка A и B са спектрално еквивалентни.

Системата (1.1.10) се решава чрез PCG алгоритъм с преобусловител $\mathcal{C}$ за A дефиниран чрез

$$\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)\,.$$

Забележка 4.1.2 *Този тип локална апроксимация за A беше използван и в т. 2.4, като разликата е в ориентацията на мрежата и необходимата структура на матрицата B. В т. 2.4 целта бе да се конструира преобуславяща матрица B с разделящи се променливи. Тук условията за устойчива* MIC(0) *факторизация са в сила за B (вж. т. 4.2) като диагоналните блокове на B позволяват ефективна паралелна реализация на получения PCG алгоритъм.*

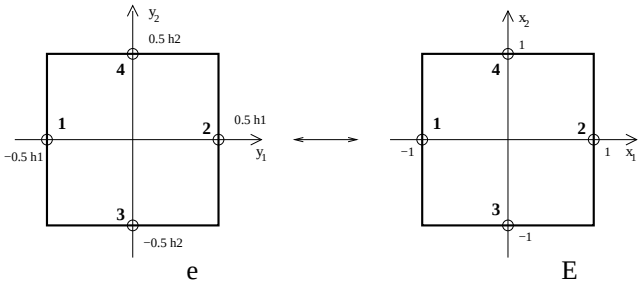
4.2 Моделен анализ на числото на обусловеност

Анализът на спектралното число на обусловеност в тази секция повтаря схемата от т. 2.4.2. Отново се разглежда анизотропната задача асоциирана с билинейната форма

$$a_h(u,v) = \sum_{e\in\omega_h} \int_e a(e)\left(\frac{\partial u}{\partial y_1}\frac{\partial v}{\partial y_1} + \varepsilon\frac{\partial u}{\partial y_2}\frac{\partial v}{\partial y_2}\right)dy\,.$$

Сега ω_h е стандартно ориентирана мрежа, състояща се от квадратни елементи със страни, успоредни на координатните оси. Коефициентът $a(e)$ е константа върху всеки елемент $e \in \omega_h$, а $\varepsilon > 0$ е параметърът на анизотропия.

Стандартната схема на МКЕ за пресмятане на елементните матрици на коравина използва възловите базисни функции $\widehat{\phi}_i$, $i = 1,\ldots,4$ (вж. т. 1.1.2) и смяна на променливите, изобразяваща базовия елемент E в $e \in \omega_h$ (Фигура 4.4), при която (за разлика от т. 2.4) не е необходима ротация.



Фиг. 4.4: Локален елемент e и базов елемент E.

Елементните матрици на коравина съответстващи на MP и MV ротирани билинейни елементи са

$$A_e^{MP} = \frac{a(e)}{3}\begin{bmatrix} 4+\varepsilon & -2+\varepsilon & -1-\varepsilon & -1-\varepsilon \\ -2+\varepsilon & 4+\varepsilon & -1-\varepsilon & -1-\varepsilon \\ -1-\varepsilon & -1-\varepsilon & 1+4\varepsilon & 1-2\varepsilon \\ -1-\varepsilon & -1-\varepsilon & 1-2\varepsilon & 1+4\varepsilon \end{bmatrix},$$

$$A_e^{MV} = \frac{a(e)}{4}\begin{bmatrix} 7+3\varepsilon & -1+3\varepsilon & -3(1+\varepsilon) & -3(1+\varepsilon) \\ -1+3\varepsilon & 7+3\varepsilon & -3(1+\varepsilon) & -3(1+\varepsilon) \\ -3(1+\varepsilon) & -3(1+\varepsilon) & 3+7\varepsilon & 3-\varepsilon \\ -3(1+\varepsilon) & -3(1+\varepsilon) & 3-\varepsilon & 3+7\varepsilon \end{bmatrix}.$$

Локалните матрици B_e се получават чрез формулите (4.1.4) и имат вида

$$B_e^{MP} = \frac{a(e)}{3}\begin{bmatrix} 2+2\varepsilon & 0 & -1-\varepsilon & -1-\varepsilon \\ 0 & 2+2\varepsilon & -1-\varepsilon & -1-\varepsilon \\ -1-\varepsilon & -1-\varepsilon & 2+2\varepsilon & 0 \\ -1-\varepsilon & -1-\varepsilon & 0 & 2+2\varepsilon \end{bmatrix},$$

$$B_e^{MV} = \frac{a(e)}{4}\begin{bmatrix} 6(1+\varepsilon) & 0 & -3(1+\varepsilon) & -3(1+\varepsilon) \\ 0 & 6(1+\varepsilon) & -3(1+\varepsilon) & -3(1+\varepsilon) \\ -3(1+\varepsilon) & -3(1+\varepsilon) & 6(1+\varepsilon) & 0 \\ -3(1+\varepsilon) & -3(1+\varepsilon) & 0 & 6(1+\varepsilon) \end{bmatrix}.$$

Тази модификация понякога се нарича „диагонална компенсация“.

За да оценим числото на обусловеност $\kappa(B^{-1}A)$ прилагаме локален анализ за A_e и B_e. Нека отбележим още веднъж, че за всяко $\varepsilon > 0$ е в сила: 1) матриците A_e и B_e са положително полуопределени; 2) $Ker(A_e) = Ker(B_e) = Span\{\mathbf{e}\}$, където $\mathbf{e}^t = (1,1,1,1)$. При тези условия, локалните спектрални задачи за вариантите MP и MV се записват във вида

$$A_e^{MP}\mathbf{w} = \lambda B_e^{MP}\mathbf{w}\,, \quad A_e^{MV}\mathbf{w} = \lambda B_e^{MV}\mathbf{w}\,, \quad \mathbf{w} \neq c\,\mathbf{e}^t\,. \tag{4.2.5}$$

Вместо (4.2.5) е достатъчно да решим редуцирани 3×3 спектрални задачи (вж. V. Eijkhout, P. Vassilevski [43]). Изследваме локалното число на обусловеност $\kappa((B_e)^{-1}A_e) = \dfrac{\lambda_{max}}{\lambda_{min}}$ за случая $0 < \varepsilon \leq 1$. Оценките при $\varepsilon > 1$ са подобни, но с $\dfrac{1}{\varepsilon}$ вместо с ε.

За MP редуцираната спектрална задача води до следното характеристично уравнение за λ:

$$(4.2.6) \qquad det \begin{bmatrix} 4+\varepsilon-\lambda(2+2\varepsilon) & -2+\varepsilon & (1-\lambda)(-\varepsilon-1) \\ -2+\varepsilon & 4+\varepsilon-\lambda(2+2\varepsilon) & (1-\lambda)(-\varepsilon-1) \\ (1-\lambda)(-\varepsilon-1) & (1-\lambda)(-\varepsilon-1) & 1+4\varepsilon-\lambda(2+2\varepsilon) \end{bmatrix} = 0.$$

Корените му са $\lambda_1 = 1$, $\lambda_2 = \dfrac{3\varepsilon}{1+\varepsilon}$, $\lambda_3 = \dfrac{3}{1+\varepsilon}$ и за $\varepsilon \in (0,1]$ е в сила $\lambda_1 < \lambda_3$, $\lambda_2 \leq \lambda_3$, т.е. $\lambda_{max} = \lambda_3$. За интервала $\varepsilon \in (0, \dfrac{1}{2})$ имаме $\lambda_2 < \lambda_1 < \lambda_3$, откъдето следва, че $\kappa((B_e^{MP})^{-1}A_e^{MP}) = \dfrac{1}{\varepsilon}$. За $\varepsilon \in (\dfrac{1}{2}, 1)$ е в сила $\lambda_1 < \lambda_2 < \lambda_3$ и следователно $\kappa((B_e^{MP})^{-1}A_e^{MP}) = \dfrac{3}{1+\varepsilon}$. Когато $\varepsilon = \dfrac{1}{2}$, се получава, че $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 2$ и от тук $\kappa((B_e^{MP})^{-1}A_e^{MP}) = 2$. Ако задачата е изотропна, т.е. $\varepsilon = 1$, то $\lambda_2 = \lambda_3 = \dfrac{3}{2}$ и $\kappa((B_e^{MP})^{-1}A_e^{MP}) = \dfrac{3}{2}$.

Характеристичното уравнение за MV

$$(4.2.7) \qquad det \begin{bmatrix} 7+3\varepsilon-\lambda(6+6\varepsilon) & -1+3\varepsilon & (1-\lambda)(-3\varepsilon-3) \\ -1+3\varepsilon & 7+3\varepsilon-\lambda(6+6\varepsilon) & (1-\lambda)(-3\varepsilon-3) \\ (1-\lambda)(-3\varepsilon-3) & (1-\lambda)(-3\varepsilon-3) & 3+7\varepsilon-\lambda(6+6\varepsilon) \end{bmatrix} = 0$$

има корени $\lambda_1 = 1$, $\lambda_2 = \dfrac{4\varepsilon}{3(1+\varepsilon)}$, $\lambda_3 = \dfrac{4}{3(1+\varepsilon)}$. За $\varepsilon \in (0,1]$ се получава $\lambda_2 < \lambda_1$, $\lambda_2 \leq \lambda_3$, т.е. $\lambda_{min} = \lambda_2$. За интервала $\varepsilon \in (0, \dfrac{1}{3})$ е в сила $\lambda_2 < \lambda_1 < \lambda_3$, откъдето следва, че $\kappa((B_e^{MV})^{-1}A_e^{MV}) = \dfrac{1}{\varepsilon}$. За $\varepsilon \in (\dfrac{1}{3}, 1)$ имаме $\lambda_2 < \lambda_3 < \lambda_1$ и сега $\kappa((B_e^{MV})^{-1}A_e^{MV}) = \dfrac{3(1+\varepsilon)}{4\varepsilon}$. Когато $\varepsilon = \dfrac{1}{3}$, се получава, че $\lambda_1 = \lambda_3 = 1$, $\lambda_2 = \dfrac{1}{3}$ и от тук $\kappa((B_e^{MV})^{-1}A_e^{MV}) = 3$. Ако задачата е изотропна, т.е. $\varepsilon = 1$, то $\lambda_2 = \lambda_3 = \dfrac{2}{3}$ и $\kappa((B_e^{MV})^{-1}A_e^{MV}) = \dfrac{3}{2}$.

Глобалната оценка за числото на обусловеност се получава от представения локален анализ. Тя следва директно от неравенствата

$$\mathbf{v}^T A \mathbf{v} = \sum_{e \in \omega_h} \mathbf{v}_e^T L_e^T A_e L_e \mathbf{v}_e \leq \lambda_{max} \sum_{e \in \omega_h} \mathbf{v}_e^T L_e^T B_e L_e \mathbf{v}_e = \lambda_{max} \mathbf{v}^T B \mathbf{v}$$

и аналогично

$$\mathbf{v}^T A\mathbf{v} \geq \lambda_{min}\mathbf{v}^T B\mathbf{v}.$$

Следователно

$$\kappa((B^{MP})^{-1}A^{MP}) \leq \frac{1}{\varepsilon}\,,\ \varepsilon \in (0,\frac{1}{2})\,, \qquad \kappa((B^{MV})^{-1}A^{MV}) \leq \frac{1}{\varepsilon}\,,\ \varepsilon \in (0,\frac{1}{3})\,,$$
$$\kappa((B^{MP})^{-1}A^{MP}) \leq \frac{3}{1+\varepsilon}\,,\ \varepsilon \in [\frac{1}{2},1]\,, \quad \kappa((B^{MV})^{-1}A^{MV}) \leq \frac{3(1+\varepsilon)}{4\varepsilon}\,,\ \varepsilon \in [\frac{1}{3},1]\,,$$

т. е. $\kappa(B^{-1}A) \leq \kappa(B_e^{-1}A_e)$.

Функциите $f(\varepsilon) = \dfrac{3}{1+\varepsilon}$ за $\varepsilon \in [\frac{1}{2},1]$ и $g(\varepsilon) = \dfrac{3(1+\varepsilon)}{4\varepsilon}$ за $\varepsilon \in [\frac{1}{3},1]$ са монотонно намаляващи в съответните интервали за ε. Следователно $f(\varepsilon) \leq f(\frac{1}{2}) = 2$ и $g(\varepsilon) \leq g(\frac{1}{3}) = 3$, т.е. $\kappa(B^{-1}A)$ е равномерно ограничено в интервала $\varepsilon \in [\frac{1}{2},1]$ за MP и при $\varepsilon \in [\frac{1}{3},1]$ за MV. Числото $\kappa(B^{-1}A)$ е неограничено за MP при $\varepsilon \in (0,\frac{1}{2})$ и за MV при $\varepsilon \in (0,\frac{1}{3})$.

Резултатът от локалния анализ се обобщава в следната теорема.

Теорема 4.2.1 *Разглежда се анизотропна елиптична гранична задача (1.1.9), дискретизирана върху квадратна мрежа от неконформни КЕ. Тогава:*

(i) разредената апроксимация B на матрицата на коравина A удовлетворява условията на Теорема 4.1.1 за устойчива MIC(0) *факторизация и за двата случая MP и MV;*

(ii) матриците B и A са спектрално еквивалентни. При това, по отношение на произволни скокове на коефициентите са в сила следните равномерни оценки за относителното число на обусловеност:

$$\kappa((B^{MP})^{-1}A^{MP}) \leq 2 \quad за \quad \varepsilon \in [\frac{1}{2},1]\,, \tag{4.2.8}$$
$$\kappa((B^{MV})^{-1}A^{MV}) \leq 3 \quad за \quad \varepsilon \in [\frac{1}{3},1]\,. \tag{4.2.9}$$

Важно е да се отбележи, че условието $\widetilde{L} \geq 0$ на **Теорема 4.1.1** за A^{MV} не е удовлетворено за никое $\varepsilon \in (0,\,1]$, а за A^{MP}, то не е в сила за $\varepsilon \in (0,\,\frac{1}{2})$. За матрицата B, то е в сила за всяко $\varepsilon \in (0,\,1]$ за MP и MV.

Изчислителната сложност на една итерация на предложения алгоритъм е

$$\mathcal{N}_{it}^{PCG/MIC(0)}(A^{-1}\mathbf{b}) \approx \mathcal{N}(\mathcal{C}^{-1}\mathbf{v}) + \mathcal{N}(A\mathbf{v}) + 10N\,.$$

Матрицата A има не повече от 7 ненулеви елемента в ред и следователно $\mathcal{N}(A\mathbf{v}) \approx 13N$. За системата с преобусловителя $\mathcal{C}$ се решават една система с долно-триъгълна матрица, една – с горно-триъгълна и една – с диагонална (вж. (4.1.2), но за матрицата B). Всеки от триъгълните множители има не повече от 3 (диагонален и два извъндиагонални) ненулеви елемента и за решаването на съответните системи чрез прав или обратен ход са необходими по $\approx 5N$ аритметични операции (ар. оп.). За системата с диагоналната матрица са необходими N ар. оп. Общо $\mathcal{N}(\mathcal{C}^{-1}\mathbf{v}) \approx 11N$, откъдето

$$\mathcal{N}_{it}^{PCG/MIC(0)}(A^{-1}\mathbf{b}) \approx 34N\,. \tag{4.2.10}$$

Скоростта на сходимост на PCG зависи от $\kappa(\mathcal{C}^{-1}A)$, където $\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)$. За моделни задачи $\kappa(\mathcal{C}^{-1}A)$ е изследвано от I. Gustafsson [51, 52] и R. Blaheta [26], като е показано, че скоростта на сходимост е с порядък $\mathcal{O}(N^{\frac{1}{4}})$, т.е. $\kappa(\mathcal{C}^{-1}A) = \mathcal{O}(N^{\frac{1}{2}})$. В множество публикации, същият порядък за броя на итерациите е експериментално потвърден за широк клас задачи включително върху неструктурирани мрежи.

4.3 Числени експерименти – I

Разгледана е моделната анизотропна задача $-u_{xx} - \varepsilon u_{yy} = f(x,y)$ в единичния квадрат, като са предположени хомогенни гранични условия на Дирихле върху долната страна на границата. Представените числени резултати илюстрират скоростта на сходимост на PCG алгоритъма за разглеждания MIC(0) преобусловител. За двата варианта MP и MV на базисните функции, се варира размерността на дискретната задача и стойността на параметъра на анизотропия $\varepsilon \in (0,1]$. Относителният критерий за спиране на итерационния процес в PCG алгоритъма е $\dfrac{(\mathcal{C}^{-1}r^{n_{it}}, r^{n_{it}})}{(\mathcal{C}^{-1}r^0, r^0)} < 10^{-6}$, където r^i е резидуала на i-тата итерация, с $\mathcal{C}$ е означен преобусловителят, а $(\cdot,\cdot)$ е стандартното Евклидово скаларно произведение. Използвана е равномерна квадратна мрежа с параметър $h = \dfrac{1}{n}$. Размерността на дискретната задача е $N = 2n(n+1)$. Полученият брой итерации за MP и MV е представен съответно в Таблица 4.1 и Таблица 4.2. Първата им колона представя мрежовия параметър чрез стойността n, а в останалите е даден броят на итерациите за различни стойности на параметъра на анизотропия ε.

От представените числени резултати можем да направим следните изводи:

- Броят на итерациите във всички случаи е $O(\sqrt{n}) = O(N^{1/4})$.
- Скоростта на сходимост е по-добра за вариант MP.

Таблица 4.1: Брой на итерациите на PCG: MIC(0) преобуславяне за MP.

$n \setminus \varepsilon$	1.0	0.8	0.6	0.55	0.51	0.5	0.49	0.45	0.4	0.2	0.1	0.01
64	34	35	37	37	38	38	38	40	40	52	64	161
128	49	48	52	54	54	54	55	56	57	73	100	262
256	71	70	73	76	76	77	77	78	81	105	141	392
512	104	99	104	106	109	108	109	110	114	148	201	624
1024	148	140	149	152	152	154	155	155	160	212	285	910

Таблица 4.2: Брой на итерациите на PCG: MIC(0) преобуславяне за MV.

$n \setminus \varepsilon$	1.0	0.8	0.6	0.4	0.35	0.34	0.33	0.32	0.3	0.2	0.1	0.01
64	39	40	42	47	49	49	50	50	50	57	70	181
128	56	58	62	68	72	73	72	72	73	85	111	293
256	81	83	87	95	100	102	103	104	106	121	166	438
512	119	118	124	136	143	145	147	150	153	174	234	697
1024	167	169	178	188	198	195	199	209	218	248	329	1050

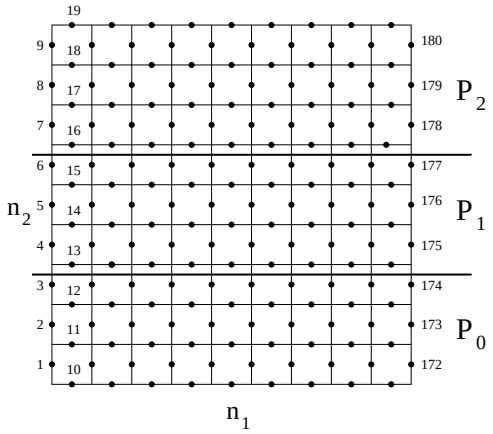
- В интервалите на равномерна ограниченост за числото на обусловеност (вж. (4.2.8), (4.2.9)) броят на итерациите се увеличава плавно с намаляване на ε.
- Когато $\kappa(B^{-1}A) \to \infty$, т.е. при $\varepsilon \to 0$, броят на итерациите бързо расте и в двата случая (MP и MV).

4.4 Паралелна реализация

Как се разпределят данните и изчисленията между процесорите? Какъв тип комуникации са необходими? Колко добър е предложеният паралелен алгоритъм? Това са основните въпроси разгледани в тази секция.

4.4.1 Данни, изчисления и комуникации

Нека с P_0, P_1, ... , P_{N_p-1} са означени процесорите на паралелната изчислителна система. Предполага се също, че моделната квадратна област показана на Фигура 4.5 е разделена на $n_1 \times n_2$ неконформни четириъгълни елементи. Всяка вертикална линия от възли (използвано е подреждане на неизвестните по стълбове) съответства на един от блоковете по диагонала на матриците A и B (вж. също Фигура 4.1). Забеляз-



Фиг. 4.5: Разпределяне на данните – $N_p = 3$, $N = n_1(2n_2 + 1) + n_2$, $n_1 = 9$, $n_2 = 9$.

ва се, че първата линия има n_2 възела, втората $(n_2 + 1)$ възела, третата има отново n_2 възела и т. н. Това води до алтернативно променяща се големина на блоковете по диагонала на A (и B), като нечетните блокове са от ред n_2, а за четните той е $(n_2 + 1)$. Общият брой блокове е $2n_1 + 1$ и размерността N на дискретната задача е $N = n_1(2n_2 + 1) + n_2$.

Областта се разделя на N_p хоризонтални ивици с приблизително равен брой елементи. Нека $n_2 = q.N_p + r$, $q, r \in \mathbb{N}$, $r < N_p$. Тогава първите r ивици имат $q + 1$ линии от елементи, останалите се състоят от q реда и всяка линия има n_1 елемента. Всеки две ивици с обща граница се асоциират с процесори с последователни номера. Възлите по границите на тези ивици принадлежат по предположение на процесора с по-голям номер (вж. Фигура 4.5). Това води до разделяне на всеки от блочните редове на матриците A и B на ивици с приблизително равен брой уравнения. Всички вектори са разпределени по същия начин. Трябва да се отбележи, че заради вертикалното подреждане на неизвестните, всеки процесор съдържа ивица от всяко от блочните уравнения, а не една ивица от цялата система.

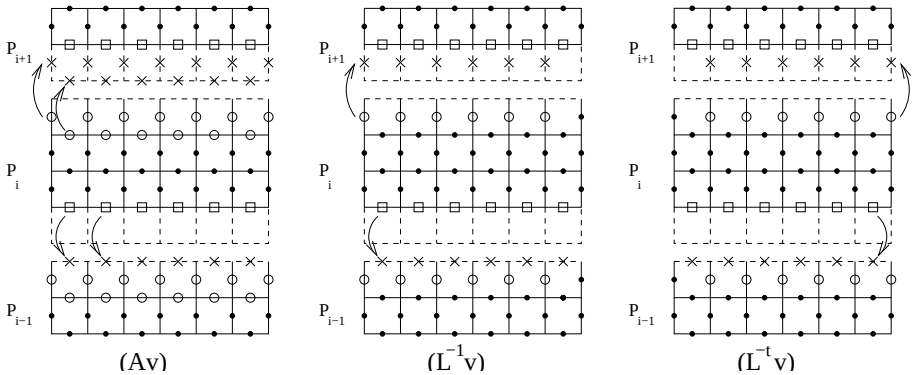
Как са разпределени изчисленията и какви комуникации са необходими? Всяка итерация на PCG (вж. т. 1.2.2) включва решаване на една система с преобусловителя $\mathcal{C}$, едно умножение на матрица по вектор с изходната матрица A, две скаларни произведения и три свързани векторни операции от вида $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$.

Всеки процесор пресмята неговата част от вектора $\mathbf{v}$ и не са необходими комуникации за свързаните векторни операции. След пресмятане на скаларните произведения съответстващи на техните части от вектори, процесорите трябва да извършат една глобална комуникация за едно число, за да сумират крайния резултат.

За да получи компонентите на $A\mathbf{v}$, за които даден процесор, например P_i, отгова-

ря, той най-напред трябва да комуникира с процесори P_{i-1} и P_{i+1}. При това P_i трябва да получи някои компоненти на вектора $\mathbf{v}$ и да изпрати на P_{i-1} и P_{i+1} съответните данни. След като приключи прехвърлянето, P_i може да извърши умножението на матрица по вектор с неговата част от A.

Фигура 4.6 илюстрира комуникациите отнасящи се за P_i, както за $A\mathbf{v}$, така и за



Фиг. 4.6: Схема на комуникациите за умножение на матрица по вектор (Av) и за решаване на системи с долно- $(L^{-1}v)$ и горно- $(L^{-t}v)$ триъгълна матрица.

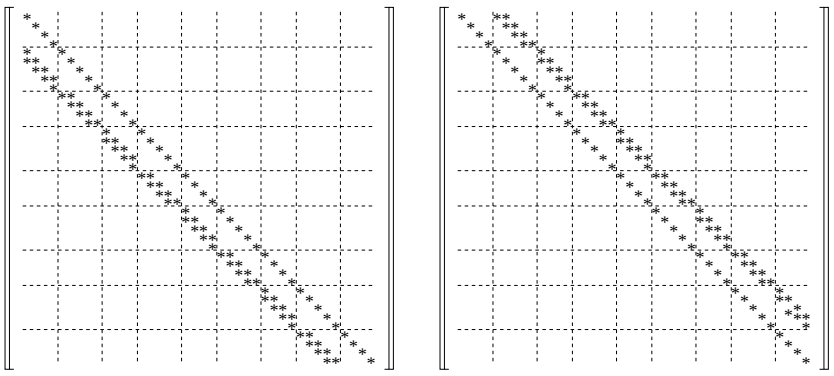
решаването на системи с преобусловителя $\mathcal{C}$. Всяка от компонентите (Av), $(L^{-1}v)$ и $(L^{-t}v)$ представлява ивицата от областта свързана с процесор P_i и най-близките части от данните в P_{i-1} и P_{i+1}. Пунктираните линии означават, че данните от тези части на областта не се намират в процесора, но е осигурена допълнителна памет, за да се извършат комуникациите. С $\circ$ са означени компонентите, които P_i изпраща на P_{i+1}, а с $\square$ – тези за P_{i-1}. Знакът $\times$ показва мястото, където се записват прехвърлените компоненти. По аналогичен начин, процесорът P_i трябва и да получи съответните данни от процесори P_{i-1} и P_{i+1}.

И така, за да се извърши умножението на матрица по вектор, процесор P_i трябва да изпрати $2n_1 + 1$ елемента от $\mathbf{v}$ (всички $\circ$-чета) на P_{i+1} и да получи от него n_1 (всички $\square$-чета на P_{i+1}) числа. Всички необходими за целта данни са пресметнати на предишната итерация. Те могат да се прехвърлят на две части – една, която да се изпрати от P_i и една, която да се получи от него. След това, P_i трябва да комуникира с P_{i-1} по подобен начин. Сега той изпраща един вектор с дължина n_1 и получава вектор с дължина $2n_1 + 1$. В случая $N_p = 2$ единият от съседите P_{i+1} и P_{i-1} няма да съществува и следователно трябва да се прехвърлят само половината от горните данни.

За да се реши системата с преобусловителя (вж. (4.1.1), (4.1.2))

$$\mathcal{C}_{\mathrm{MIC}(0)}(B)\mathbf{w} \equiv (X - \widetilde{L})X^{-1}(X - \widetilde{L})^t\mathbf{w} = \mathbf{v}$$

трябва да се извършат следните три стъпки: 1) намиране на $\mathbf{y}$ от $L\mathbf{y} = \mathbf{v}$, където $L = X - \widetilde{L}$; 2) пресмятане на $\mathbf{y} := X\mathbf{y}$; и 3) намиране на $\mathbf{w}$ от $L^t\mathbf{w} = \mathbf{y}$. Ненулевата структура на L и L^t е илюстрирана на Фигура 4.7. Матрицата X е диагонална,



L L^t

Фиг. 4.7: Ненулева структура на триъгълните множители L и L^t на преобусловителя $\mathcal{C}_{\mathrm{MIC}(0)}(B)$ за A.

разпределена между процесорите по описания по-горе начин. Важното предимство на матрицата B е, че всичките ѝ блокове по главния диагонал са диагонални. В този случай, $\widetilde{L}$ има нулеви блокове $\widetilde{L}_{ii}$, $i = 1, \ldots, 2n_1+1$, което води до диагонални блокове $L_{ii} = (X - \widetilde{L})_{ii}$. Системата $L\mathbf{y} = \mathbf{v}$ се решава чрез стандартния прав ход. Блокът L_{11} е диагонален, разделен на N_p ивици – по една за всеки процесор. Така $L_{11}\mathbf{y}_1 = \mathbf{v}_1$ се решава паралелно без комуникации и с балансирани по равно аритметични операции. След това трябва да се определи $\mathbf{y}_2$ от $L_{21}\mathbf{y}_1 + L_{22}\mathbf{y}_2 = \mathbf{v}_2$. Компонентата $\mathbf{y}_1$ е пресметната на предишната стъпка, но всеки процесор съдържа ивица от $\mathbf{y}_1$. Блокът L_{21} е двудиагонален и следователно процесорът P_i трябва да изпрати една компонента на P_{i+1} и да получи друга от P_{i-1} (вж. Фигура 4.6($L^{-1}v$)). Тогава $L_{22}\mathbf{y}_2 = \mathbf{v}_2 - L_{21}\mathbf{y}_1$ се решава паралелно без други комуникации. След това трябва да се реши $L_{32}\mathbf{y}_2 + L_{33}\mathbf{y}_3 = \mathbf{v}_3$. Отново трябва да се прехвърли една компонента на $\mathbf{y}_2$, но този път на процесор P_{i-1}. Процедурата продължава докато се намери и последния блок на $\mathbf{y}$. Втората стъпка $\mathbf{y} := X\mathbf{y}$ е без комуникации и е напълно паралелна. Последната трета стъпка се извършва по подобен на стъпка 1) начин, използвайки стандартния обратен ход. Комуникациите са илюстрирани на Фигура 4.6($L^{-t}v$). Когато решаваме задачата върху два процесора, данните за прехвърляне на стъпки 1) и 3) са отново два пъти по-малко. Важно е да се отбележи, че тук не могат да се групират компоненти и да се прехвърлят по няколко на веднъж. В този смисъл все още има рекурсия, но тя е за блоковете, а всеки блок се обработва паралелно.

4.4.2 Оценки за паралелни времена и ускорения

Вече могат да се изведат оценки за паралелните времена при предположенията направени в т. 1.3.2. Времето T_{N_p} за решаване на системата (1.1.10) с N_p процесора зависи от изчислителната сложност, типа и количеството необходими комуникации и характеристиките на конкретния паралелен компютър. Тъй като паралелните свойства не зависят от броя на итерациите, достатъчно е да се оцени времето T_{N_p} за една итерация и да се използва при анализа на ускорението и ефективността. Изчислителната сложност за процесор е $\mathcal{N}_{\rm PCG}^{it,N_p} \approx \dfrac{34N}{N_p}$ защото: решаване на една система с $\mathcal{C}_{N\times N}$ изисква $\approx 11\dfrac{N}{N_p}$ ар. оп.; едно умножение на матрица по вектор с $A_{N\times N}$ се извършва чрез $\approx 13\dfrac{N}{N_p}$ ар. оп.; две скаларни произведения се пресмятат за $4\dfrac{N}{N_p}$ ар. оп.; и за трите свързани векторни операции $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$ са необходими $6\dfrac{N}{N_p}$ ар. оп. Следователно, времето за изчисления е

$$T_a^{it} = 34\frac{N}{N_p}.t_a\,. \tag{4.4.11}$$

Времето за комуникации е сума от съответните времена за скаларните произведения, умножението на матрица с вектор и решаването на системата с преобусловителя. За всяко от скаларните произведения е необходима глобална комуникация, съставена от един all_to_one reduce за едно число и един one_to_all broadcast отново за едно число: $T_{com}(in.\ pr.) = r(N_p,\,1) + b(N_p,\,1) = 2\,b(N_p,\,1)$ (вж. т. 1.3.2). Съгласно Таблица 1.1, общото време $T_{com}(2\,in.\ pr.) = 4\,b(N_p,\,1)$ за двете скаларни произведения за пръстен, квадратна мрежа и хиперкуб (означени съответно с горни индекси r, m и h) е:

$$\begin{aligned}
T_{com}^{r}(2\,in.\ pr.) &= 4\,(t_s + t_w)\left\lceil\frac{N_p}{2}\right\rceil\,,\\
T_{com}^{m}(2\,in.\ pr.) &= 8\,(t_s + t_w)\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil\,,\\
T_{com}^{h}(2\,in.\ pr.) &= 4\,(t_s + t_w)\log N_p\,.
\end{aligned}$$

Тези времена зависят от архитектурата и броя на процесорите, но не зависят от размерността на задачата. Следователно можем да предположим, че те не влияят на водещите членове в общите паралелни времена, когато $n_1 \gg N_p$. В разглеждания случай, комуникациите за $A\mathbf{v}$ и $\mathcal{C}^{-1}\mathbf{v}$ са между процесори с последователни индекси. Те са локални ако процесорите са съседи и физически. Следователно времето за

комуникации за $A\mathbf{v}$ се оценява с $T_{com}(A\mathbf{v}) = 2t_s + (3n_1+1)t_w$, при $N_p = 2$ и $T_{com}(A\mathbf{v}) = 4t_s + 2(3n_1+1)t_w$, при $N_p > 2$. За решаването на системата с преобусловителя времето е $T_{com}(\mathcal{C}^{-1}\mathbf{v}) = 4n_1(t_s + t_w)$ за случая $N_p = 2$ и $T_{com}(\mathcal{C}^{-1}\mathbf{v}) = 8n_1(t_s + t_w)$ за случая $N_p > 2$ ($T_{com}(\mathcal{C}^{-1}\mathbf{v}) = T_{com}(L^{-1}\mathbf{v}) + T_{com}(L^{-t}\mathbf{v})$).

Така общото време за комуникации на итерация (когато $n_1 \gg N_p$) има вида

$$T_{com}^{it} \approx \begin{cases} 4n_1.t_s + 7n_1.t_w\,, & \text{при } N_p = 2\,, \\ 8n_1.t_s + 14n_1.t_w\,, & \text{при } N_p > 2\,. \end{cases} \tag{4.4.12}$$

Следователно, ако $N_p > 2$, времето за една PCG итерация е

$$T_{N_p}^{it} = T_a^{it} + T_{com}^{it} \approx 34\frac{n_1(2n_2+1)+n_2}{N_p}.t_a + 8n_1.t_s + 14n_1.t_w\,. \tag{4.4.13}$$

Какво може да се очаква за реалното време за изпълнение? Поведението на времето за комуникации не трябва да зависи от броя на процесорите. Единственото изключение е случаят $N_p = 2$, когато то трябва да е два пъти по-малко. По-малките параметри на компютъра t_s и t_w намаляват времето за комуникации и водят в общия случай до по-добри ускорение и ефективност. В частност, при паралелен компютър с обща памет, на практика не се извършват комуникации. Това означава, че ускорението $S_{N_p} = \dfrac{T_1}{T_{N_p}}$ в този случай ще бъде близко до теоретичната си горна граница $S_{N_p} \leq N_p$. Това е основното предимство на предложения алгоритъм. Недостатък е коефициента n_1 пред стартовото време t_s. Големи отношения t_s/t_w и t_s/t_a могат да доведат до намаляване на производителността с повече от 50% за големи дискретни задачи.

4.5 Числени експерименти – II

Компютърната програма реализираща описания паралелен алгоритъм е разработена на C използвайки стандарта MPI. Анализирана е получената производителност върху Parmac и Thea – два Beowulf клъстера с Linux операционна система и характеристики, описани в т. 1.3.3.

Отново разглеждаме моделно уравнение на Поасон в единичния квадрат (вж. т. 4.3). Ускорението и ефективността за паралелните алгоритми, съответстващи на вариантите MP и MV, са представени в Таблица 4.3 и Таблица 4.4. И двете таблици са с еднаква структура. Има две числа във всяка клетка на първата колона – броят n на елементите по всяко направление и съответният брой на итерациите за получаване на решението за тази размерност на задачата. Броят на процесорите е даден във

Таблица 4.3: Паралелен PCG/MIC(0), Алгоритъм MP.

		Parmac			Parmac(N)			Thea		
n, it	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	1	16.74			16.82			9.26		
256	2	10.73	1.56	0.78	14.83	1.13	0.57	9.48	0.98	0.49
71	4	11.71	1.43	0.36	17.16	0.98	0.25	11.83	0.78	0.20
	8	10.92	1.53	0.19	20.07	0.84	0.11	11.08	0.84	0.11
	1	99.42			99.81			54.02		
512	2	63.69	1.56	0.78	68.71	1.45	0.73	41.11	1.31	0.66
104	4	50.22	1.98	0.50	65.44	1.53	0.38	41.14	1.31	0.33
	8	37.74	2.63	0.33	64.24	1.55	0.19	35.34	1.53	0.19
	1	577.00			567.11			288.27		
1024	2	373.19	1.55	0.78	340.12	1.67	0.84	198.70	1.45	0.73
148	4	233.55	2.47	0.62	274.27	2.07	0.52	153.35	1.88	0.47
	8	171.28	3.37	0.42	217.27	2.61	0.33	124.45	2.32	0.29

втората колона. Останалите колони са групирани по три – една група за всеки начин на изпълнение на програмата (Parmac и Parmac(N) се различават по опциите на „mpirun“ – вж. т. 1.3.3). Всяка от тези групи има по три полета – в първото е дадено измереното cpu-време в секунди, а останалите две представят ускорението $S_{N_p} = \frac{T_1}{T_{N_p}}$ и ефективността $E_{N_p} = \frac{S_{N_p}}{N_p}$. Времената T_{N_p} са най-добрите, получени от три измервания.

Изводите от тези таблици са както следва:

- За дадена „машина“: cpu-времената за MV са по-големи от тези за MP заради по-големия брой итерации; ускорението и ефективността са приблизително равни за MP и MV, което потвърждава, че свойствата на паралелния алгоритъм не зависят от типа на базисните функции.

- За даден брой процесори: cpu-времената за $N_p = 1$ на Thea са приблизително два пъти по-малки от тези на Parmac и за двата начина на изпълнение; ускорението и съответно ефективността се увеличават с нарастване на размерността на задачата с едно изключение – случаят $N_p = 2$ за Parmac.

- За дадена размерност на задачата: общият извод е, че на Parmac, когато се използва обща памет, ускоренията са най-добри (освен случая $N_p = 2, n = 1024$), а на Thea те са най-малки;

Таблица 4.4: Паралелен PCG/MIC(0), Алгоритъм MV.

		Parmac			Parmac(N)			Thea		
n, it	N_p	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}	T_{N_p}	S_{N_p}	E_{N_p}
	1	19.09			19.16			10.55		
256	2	12.24	1.56	0.78	16.93	1.13	0.57	10.79	0.98	0.49
81	4	13.32	1.43	0.36	19.49	0.98	0.25	13.43	0.79	0.20
	8	12.93	1.48	0.19	22.85	0.84	0.11	13.04	0.81	0.10
	1	113.45			114.09			63.17		
512	2	73.00	1.55	0.78	78.49	1.45	0.73	49.91	1.35	0.68
119	4	57.66	1.97	0.49	77.14	1.48	0.37	47.04	1.34	0.34
	8	42.51	2.67	0.33	82.25	1.39	0.17	41.87	1.51	0.19
	1	635.63			639.18			326.63		
1024	2	409.39	1.55	0.78	383.58	1.67	0.84	223.35	1.46	0.73
167	4	264.88	2.40	0.60	309.19	2.07	0.52	173.26	1.89	0.47
	8	175.63	3.62	0.45	245.17	2.61	0.33	140.73	2.32	0.29

Когато анализираме получените резултати, трябва да обърнем внимание на някои особености на двата клъстера:

- Процесорите на Thea са по-бързи и затова времената са по-малки в сравнение с тези на Parmac.

- Когато е възможно да се използва обща памет на Parmac, времето за комуникации е по-малко и съответно общите cpu-времена са по-малки от тези, при които е използвана опция N в „mpirun“.

- За относително малки задачи, общото време върху Parmac се влияе само от времето за комуникации. От друга страна, когато размерността е по-голяма, двата процесора от един възел се състезават за достъп до паметта и дори комуникацията да отнема по-малко време отколкото ако програмата се изпълнява на два възела, общото време е по-голямо. Това е причината за приблизително равните ускорения на Parmac за $N_p = 2$, и за по-доброто ускорение на Parmac(N) отколкото на Parmac за $N_p = 2$ и $n = 1024$.

Как параметрите на съответните изчислителни системи влияят върху поведението на предложените и изследвани алгоритъм и програмна реализация? На този въпрос не може да се отговори само въз основа на общите cpu-времена. От тези

таблици не може да се види и какво е конкретното поведение на времената за изчисления и комуникации. Също така не може да се каже дали теоретичните оценки в предишната секция добре представят основните свойства на паралелния алгоритъм. В Таблица 4.5 и Таблица 4.6 е представено разпределението на измерените времена

Таблица 4.5: Разпределение на времето, Алгоритъм MV, $n = 512$, $N_{it} = 119$.

машина	N_p	E_{N_p}	T_{N_p}	T_a	T_c	T_c^{u*v}	T_c^{M*v}	T_c^{prec}
	1		109.85	109.85				
Parmac	2	0.77	71.63	70.35	1.28	0.02	0.08	1.18
	4	0.48	57.44	35.60	21.84	0.43	0.04	21.37
	8	0.32	43.25	18.17	25.08	1.26	0.06	23.76
	1		113.12	113.12				
Parmac(N)	2	0.73	77.48	55.82	21.66	0.04	0.29	21.33
	4	0.37	75.88	27.72	48.16	1.07	0.26	46.83
	8	0.19	73.77	16.78	56.99	1.32	0.31	55.36
	1		60.29	60.29				
Thea	2	0.66	45.89	30.55	15.34	0.14	0.23	14.97
	4	0.32	46.74	16.10	30.64	0.23	0.21	30.20
	8	0.18	42.76	8.74	34.02	0.30	0.20	33.52

за изчисления и комуникации за Алгоритъм MV съответно за $n = 512$ и $n = 1024$. Първите три колони дават информация за машината, броя на процесорите и получената ефективност. Измерените времена са показани в останалата част от колоните. Тук T_{N_p} е общото cpu-време за решаване на задачата с N_p процесора, T_a и T_c са времената съответно за изчисления и комуникации. Следвайки теоретичните оценки, комуникациите са разделени по тип – за скаларни произведения T_c^{u*v}, за умножение на матрица по вектор T_c^{M*v} и за решаване на системата с преобусловителя T_c^{prec}. Изчисленията и комуникациите не се припокриват, тъй като са използвани блокиращи SEND и RECEIVE операции на MPI. Следователно в сила са следните равенства:

$$T_{N_p} = T_a + T_c, \qquad T_c = T_c^{u*v} + T_c^{M*v} + T_c^{prec}.$$

Вижда се, че времето за изчисления на всички машини намалява почти два пъти при удвояване на броя на процесорите. Изключенията за Parmac се обясняват със състезанието за достъп до паметта при големи задачи. Както трябва да се очаква, забавянето е резултат от комуникациите. Предвижданото, че комуникациите за скаларните произведения и умноженията на матрица по вектор ще са пренебрежими

Таблица 4.6: Разпределение на времето, Алгоритъм MV, $n = 1024$, $N_{it} = 167$.

машина	N_p	E_{N_p}	T_{N_p}	T_a	T_c	T_c^{u*v}	T_c^{M*v}	T_c^{prec}
	1		637.30	637.30				
Parmac	2	0.77	412.62	407.71	4.91	0.10	0.23	4.58
	4	0.60	264.21	201.20	63.01	0.37	0.16	62.48
	8	0.47	170.18	100.64	69.54	5.07	0.22	64.25
	1		638.94	638.94				
Parmac(N)	2	0.84	381.55	319.48	62.07	0.14	0.73	61.20
	4	0.52	308.14	157.79	150.35	4.63	0.67	145.05
	8	0.32	247.03	96.96	150.07	5.19	0.83	144.05
	1		324.81	324.81				
Thea	2	0.73	223.95	171.27	52.68	0.93	3.02	48.73
	4	0.47	172.92	84.54	88.38	0.84	0.81	86.73
	8	0.29	140.87	44.51	96.36	0.81	0.58	94.97

в сравнение с комуникациите за решаване на системата с преобусловителя също се потвърждава. Поради по-бързите процесори, времената за изчисление на Thea са два пъти по-малки от тези на Parmac, но отношението за комуникациите не е същото. Ето защо ефективността на Thea е по-малка като цяло. Има огромна разлика между времената T_c за $N_p = 2$ върху Parmac и Parmac(N). За комуникациите при Parmac е използвана обща памет срещу TCP/IP за Parmac(N). Това показва потенциала на предложения паралелен алгоритъм и разработената програма за реализации върху системи с обща памет. Очакването, че времето за комуникации няма да зависи от броя на процесорите също се потвърди – то е приблизително едно и също за $N_p = 4$ и $N_p = 8$ за всички машини. Случаят $N_p = 2$ е анализиран отделно. За неговото време T_c се предполагаше, че ще е два пъти по-малко отколкото това за $N_p = 4$, което се потвърждава от експеримента в случая на разпределена памет.

Направеният анализ показва, че изследванията за по-нататъшно подобряване на ефективността на паралелната реализация, могат да се насочат към:

- За машини с разпределена памет може локално да се пренаредят изчисленията при решаване на системата с преобусловителя и да се използват неблокиращи SEND и RECEIVE операции на MPI, за да се позволи припокриване на изчисленията и комуникациите.

- За машини с обща памет може да се използва Open MP или комбинация на

MPI и Open MP за клъстери от компютри с обща памет.

Литература

[1] А. Андреев, Хр. Джиджев, Бл. Сендов, Н. Янев, *Обзор по паралелно смятане,* БАН, София, 1985.

[2] Б. Н. Парлет, *Симетричная проблема собственных значений, Численные методы,* Москва, „Мир“, 1983.

[3] А. А. Самарский, *Теория разностных схем,* Москва, „Наука“, 1983.

[4] Бл. Сендов, В. Попов, *Числени методи I, II,* София, „Наука и изкуство“, 1978.

[5] Г. Стренг, Дж. Фикс, *Теория метода конечных элементов,* Москва, „Мир“, 1977.

[6] К. Хорстман, *Принципи на програмирането със C++,* ИК СОФТЕХ, София, 2000.

[7] Х. Шилдт, *C – практически самоучител,* СофтПрес ООД, София, 2001.

[8] T. Arbogast, Z. Chen, On the implementation of mixed methods as nonconforming methods for second order elliptic problems, *Math. Comp.,* **64** (1995), 943–972.

[9] D. N. Arnold, F. Brezzi, Mixed and nonconforming finite element methods: implementation, postprocessing and error estimates, *RAIRO, Model. Math. Anal. Numer.,* **19** (1985), 7–32.

[10] E. I. Atanassov, Measuring the performance of a Power PC cluster, *Computational Science – ICCS 2002, Springer LNCS,* **2330** (2002), 628–634.

[11] O. Axelsson, *Iterative solution methods,* Cambridge, University press, 1996.

[12] O. Axelsson, I. Faragó, J. Karátson, Sobolev space preconditioning for Newton's method using domain decomposition, *Numer. Lin. Alg.* **9** (2002).

[13] A. Banegas, Fast Poisson solvers for problems with sparsity, *Math. Comp.,* **32** (1978), 441–446.

[14] R. Bank, Marching algorithms for elliptic boundary value problems. II: The variable coefficient case, *SIAM J. Numer. Anal.,* **14** (1977), 950–970.

[15] R. Bank, D. Rose, An $\mathcal{O}(n^2)$ method for solving constant coefficient boundary value problems in two dimensions, *SIAM J. Numer. Anal.,* **12** (1975), 529–540.

[16] R. Bank, D. Rose, Marching algorithms for elliptic boundary value problems. I: The constant coefficient case, *SIAM J. Numer. Anal.,* **14** (1977), 792–829.

[17] G. Bencheva, Comparative performance analysis of 2D separable elliptic solvers, *Proceedings of the XIII-th summer school „Software and Algorithms of Numerical Mathematics“,* Nečtiny, Czech Republic, (1999), 11–27.

[18] G. Bencheva, Comparative analysis of marching algorithms for separable elliptic problems, *Numerical Analysis and its Applications, Springer LNCS,* **1988** (2001), 76–83.

[19] G. Bencheva, MPI parallel implementation of a fast separable solver, *Large-Scale Scientific Computing, Springer LNCS,* **2179** (2001), 454–461.

[20] G. Bencheva, *Parallel implementation of a generalized marching algorithm,* Technical Report ISC-04-10-MATH (2004), (`http://www.isc.tamu.edu/iscpubs/0410.ps`).

[21] G. Bencheva, Parallel performance comparison of three direct separable elliptic solvers, *Large-Scale Scientific Computing, Springer LNCS,* **2907** (2004), 421–428, (Extended version: Technical Report ISC-03-02-MATH (`http://www.isc.tamu.edu/iscpubs/0302.ps`).)

[22] G. Bencheva, I. Georgiev, S. Margenov, Two-lewel preconditioning of Crouzeix-Raviart anisotropic FEM systems, *Large-Scale Scientific Computing, Springer LNCS,* **2907** (2004), 76–84.

[23] G. Bencheva, S. Margenov, On a preconditioning strategy for rotated linear FEM elliptic systems, *Proceedings, PRISM'01,* University of Nijmegen, The Netherlands, (2001), 87-90.

[24] G. Bencheva, S. Margenov, Parallel incomplete factorization preconditioning of rotated linear FEM systems, *Journal of Computational and Applied Mechanics,* **4(2)** (2003), 105–117.

[25] G. Bencheva, S. Margenov, Performance analysis of a parallel MIC(0) preconditioning of rotated bilinear nonconforming FEM systems, *Mathematica Balkanica,* **17** (2003), 319–335.

[26] R. Blaheta, Displacement decomposition – incomplete factorization preconditioning techniques for linear elasticity problems, *Numer. Linear Algebra Appl.,* **1** (1994), 107–126.

[27] R. Blaheta, S. Margenov, M. Neytcheva, Uniform estimate of the constant in the strengthened CBS inequality for anisotropic non-conforming FEM systems, *Numer. Linear Algebra Appl.,* **11** (2004), 309–326.

[28] J. Bramble, R. Ewing, J. Pasciak, A. Schatz, A preconditioning technique for the efficient solution of problems with local grid refinement, *Comput. Meth. Appl. Mech. Eng.,* **67** (1988), 149–159.

[29] J. Bramble, J. Pasciak, A. Schatz, An iterative method for elliptic problems on regions partitioned into substructures, *Math. Comp.,* **46** (1986), 361–370.

[30] F. Brezzi, M. Fortin, *Mixed and hybrid finite element methods*, Springer-Verlag, New York, Berlin, Heidelberg, 1991.

[31] L. Briggs, T. Turnbull, Fast Poisson solvers for MIMD computers, *Parallel Comput.,* **6** (1988), 265–274.

[32] O. Buneman, A compact non-iterative Poisson solver, Tech. Report 294, Stanford University Institute of Plasma Research, Stanford, California, 1969.

[33] R.H. Chan, T.F. Chan, Circulant preconditioners for elliptic problems, *Journal of Numerical Linear Algebra with Applications,* **1(1)** (1992), 77–101.

[34] Z. Chen, Analysis of mixed methods using conforming and nonconforming finite element methods, *RAIRO, Math. Model. Numer. Anal.,* **27** (1993), 9–34.

[35] E. Chow, A priori sparsity patterns for parallel sparse approximate inverse preconditioners, *SIAM Journal on Scientific Computing,* **21(5)** (2000), 1804–1822 (`http://www.llnl.gov/CASC/linear_solvers/pubs.html`).

[36] E. Chow, Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns, *Int'l J. High Perf. Comput. Appl.,* **15** (2001), 56–74 (`http://www.llnl.gov/CASC/linear_solvers/pubs.html`).

[37] J. W. Cooley, J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Math. Comp.,* **19** (1965), 297–301.

[38] St. McCormick, *Multilevel adaptive methods for partial differential equations,* SIAM, 1989.

[39] S. McCormic, J. Thomas, The fast adaptive composite grid (FAC) method for elliptic equations, *Math. Comp.,* **46** (1986), 439–456.

[40] M. Crouzeix, P.-A. Raviart, Conforming and non-conforming finite element methods for solving the stationary Stokes equations, *RAIRO Anal. Numér.* **7 R-3** (1973), 33–76.

[41] J. Dongarra, I. Duff, D. Sorensen, H. van der Vorst, *Numerical linear algebra for high-performance computers,* SIAM, 1998.

[42] C. C. Douglas, G. Haase, U. Langer, *A tutorial on elliptic PDE solvers and their parallelization,* SIAM, 2003.

[43] V. Eijkhout, P. Vassilevski, The role of the strengthened C.B.S. inequality in multilevel methods, *SIAM Review,* **33** (1991), 405–419.

[44] R. Ewing, R. Lazarov, P. Vassilevski, Local refinement techniques for elliptic problems on cell-centered grids; II. Optimal order two-grid iterative methods, *Numer. Lin. Alg. Appl.,* **1** (1993), 1–7.

[45] R.S. Falk, Nonconforming finite element methods for the equations of linear elasticity, *Mathematics of Computation,* **57(196)** (1991), 529–550.

[46] I. Faragó, J. Karátson, *Numerical solution of nonlinear elliptic poblems via preconditioning operators. Theory and applications,* NOVA Science, 2002.

[47] M. Fortin, M. Soulie, A non-conforming piece-wise quadratic finite element on triangles, *Internat. J. Numer. Methods Engrg.,* **19** (1983), 505–520.

[48] I. Foster, *Designing and building parallel programs,* Addison-Wessley, 1995.

[49] E. Gallopoulos, Y. Saad, A parallel block cyclic reduction algorithm for the fast solution of elliptic equations, *Parallel Comput.,* **10** (1989), 143–159.

[50] G. H. Golub, C. F. Van Loan, *Matrix computations,* The Johns Hopkins University Press, Baltimore and London, 1989.

[51] I. Gustafsson, Stability and rate of convergence of modified incomplete Cholesky factorization methods, Report 79.02R. Dept. of Comp. Sci., Chalmers University of Technology, Goteborg, Sweden, 1979.

[52] I. Gustafsson, Modified incomplete Cholesky (MIC) factorization, in: D.J. Evans, ed., *Preconditioning Methods; Theory and Applications,* (Gordon and Breach Publishers, New York-London-Paris, 1983), 265–293.

[53] I. Gustafsson and G. Lindskog, On parallel solution of linear elasticity problems: Part I: Theory, *Numer. Linear Algebra Appl.,* **5** (1998), 123–139.

[54] I. Gustafsson and G. Lindskog, On parallel solution of linear elasticity problems. Part II: Methods and some computer experiments, *Numer. Linear Algebra Appl.,* **9** (2002), 205–221.

[55] W. Hackbusch, *Elliptic differential equations. Theory and numerical treatment,* Springer Series in Computational Mathematics, 18, Springer-Verlag, 1992.

[56] P. Hansbo, M.G. Larson, A simple nonconforming bilinear element for the elasticity problem, *Trends in Computational Structural Mechanics,* W.A. Wall, K.U.Bletzinger, and K. Schweizerkopf (Eds.), CIMNE, (2001), 317–327.

[57] R. Heathfield, L. Kirby et al., *C unleashed,* SAMS Publishing, USA, 2000.

[58] D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.,* **13(4)** (1976), 484–496.

[59] N. J. Higham, *Handbook of writing for the mathematical sciences,* SIAM, 1998.

[60] R. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, *J. Assoc. Comput. Mach.,* **12** (1965), 95–113.

[61] R. Hockney, The potential calculation and some applications, *Meth. Comp. Physics,* Academic Press, New York, (1969), 136–212.

[62] R. Hockney, Characterizing computers and optimizing the FACR(l) Poisson solver on parallel unicomputers, *IEEE Trans. Comput.,* **32** (1983), 933–941.

[63] C. Johnson, *Numerical solution of partial differential equations by the finite element method,* Cambridge University Press, 1987.

[64] V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to parallel computing: design and analysis of algorithms,* Benjamin-Cummings Addison-Wesley Publishing Company, Inc., 1994.

[65] Y. Kuznetsov, Numerical methods in subspaces, *Vychislitel'nye Processy i Systemy II, (G. I. Marchuk, ed.),* Nauka, Moscow, 1985, 265–350 (in Russian).

[66] Y. Kuznetsov, Block relaxation methods in subspaces, their optimization and application, *Sov. J. Numer. Anal. Math. Model.,* **4** (1989), 433–452.

[67] Y. Kuznetsov, A. M. Matsokin, On partial solution of systems of linear algebraic equations, *Vychislitel'nye Metody Lineinoi Algebry (Ed. G.I. Marchuk),* Vychisl. Tsentr Sib. Otdel. Akad. Nauk SSSR, Novosibirsk, (1978), pp. 62–89. In Russian, English translation in: *Sov. J. Numer. Anal. Math. Modelling,* **4** (1989), 453–468.

[68] R. Lazarov, S. Margenov, On a two-level parallel MIC(0) preconditioning of Crouzeix-Raviart non-conforming FEM systems, *Numerical Methods and Applications, Springer LNCS,* **2542** (2003), 191–200.

[69] I. Lirkov, S. Margenov, On circulant preconditioning of Elliptic problems in L-shaped domain, *Advances in Numerical Methods and Applications, World Scientific,* (1994), 136–145.

[70] I. Lirkov, S. Margenov, M. Paprzycki, Benchmarking performance of parallel computers using a 2D elliptic solver, *Recent advances in numerical methods and applications,* World Scientific, (1999), 491–499.

[71] I. Lirkov, S. Margenov, P. Vassilevski, Circulant block-factorization preconditioners for elliptic problems, *Computing,* **53(1)** (1994), 59–74.

[72] I. Lirkov, S. Margenov, L. Zikatanov, Circulant block-factorization preconditioning of anisotropic elliptic problems, *Computing,* **58(3)** (1997), 245–258.

[73] C. van Loan, *Computational frameworks for the fast Fourier transform,* SIAM, Philadelphia (1992).

[74] C.G. Page, *Professional programmer's guide to Fortran77,* University of Leicester, UK, 1995.

[75] Sv. Petrova, Parallel implementation of fast elliptic solver, *Parallel Computing,* **23** (1997), 1113–1128.

[76] W. Proskurowski, Numerical solution of Helmholtz equation by implicit capacitance matrix method, *ACM Trans. Math. Software,* **5** (1979), 36–49.

[77] W. Proskurowski, P. S. Vassilevski, Preconditioning capacitance matrix problems in domain imbedding, *SIAM J. Sci. Comput.,* **15** (1994), 77–88.

[78] R. Rannacher, S. Turek, Simple nonconforming quadrilateral Stokes element, *Numerical Methods for Partial Differential Equations,* **8(2)** (1992), 97-112.

[79] T. Rossi, *Fictitious domain methods with separable preconditioners*, PhD thesis, University of Jyväskylä, Department of mathematics, report 69, 1995.

[80] T. Rossi, J. Toivanen, A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension, *SIAM J. Sci. Comput.,* **20(5)** (1999), 1778–1796.

[81] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publishing Company, Boston, 1996.

[82] Y. Saad, M. Schultz, Data communication in parallel architectures, *Parallel Computing,* **11** (1989), 131–150.

[83] A. H. Sameh, S. C. Chen, D. J. Kuck, Parallel Poisson and biharmonic solvers, *Computing,* **17** (1976), 219–230.

[84] B. Smith, P. Bjorstad, W. Gropp, *Domain decomposition, parallel multilevel methods for elliptic partial differential equations,* Cambridge University Press, 1996.

[85] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, J. J. Dongarra, *MPI. The complete reference,* The MIT Press, Massachusetts, 1996.

[86] P. Swarztrauber, A direct method for the discrete solution of separable elliptic equations, *SIAM J. Numer. Anal.,* **11** (1974), 1136–1150.

[87] P. Swarztrauber, The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle, *SIAM Rev.,* **19** (1977), 490–501.

[88] P. Swarztrauber, R. Sweet, Vector and parallel methods for the direct solution of Poisson's equation, *J. Comput. Appl. Math.,* **27** (1989), 241–263.

[89] R. Sweet, A generalized cyclic reduction algorithm, *SIAM J. Numer. Anal.,* **11** (1974), 506–520.

[90] R. Sweet, A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimensions, *SIAM J. Numer. Anal.,* **14** (1977), 706–720.

[91] R. Sweet, A parallel and vector variant of the cyclic reduction algorithm, *SIAM J. Sci. Statist. Comput.,* **9** (1988), 761–765.

[92] R. Sweet, W. L. Briggs, S. Oliveira, J. L. Porsche, T. Turnbull, FFTs and three-dimensional Poisson solvers for hypercubes, *Parallel Comput..* **17** (1991), 121–131.

[93] P. S. Vassilevski, Fast algorithm for solving a linear algebraic problem with separable variables, *Compt. rend. de l'Acad. Bulg. Sci.,* **37(3)** (1984), 305–308.

[94] P. S. Vassilevski, An optimal stabilization of marching algorithm, *Compt. rend. de l'Acad. Bulg. Sci.,* **41(7)** (1988), 29–32.

[95] P. S. Vassilevski, S. Petrova, A note on construction of preconditioners in solving 3D elliptic problems by substructuring, *C. R. Acad. Bulg. Sci.,* **41(7)** (1988), 33–36.

[96] P. S. Vassilevski, S. Petrova, R. Lazarov, Finite difference schemes on triangular cell-centered grids with local refinement, *SIAM J. Sci. Stat. Comp.,* **13** (1992), 1287–1313.