VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Paralelní algoritmy pro řešení úloh pružnosti

Disertační práce

2003 Jiří Starý

Děkuji mému školiteli Prof. RNDr. Radimovi Blahetovi, CSc. za seznámení s numerickými metodami pro řešení úloh pružnosti, za odborné vedení a péči, kterou mi věnoval.

Rovněž bych chtěl poděkovat všem pracovníkům Střediska aplikované matematiky Ústavu geoniky AV ČR za jejich podporu, komentáře i povzbuzování při psaní této práce.

6. května 2003

Obsah

Anotace

V předložené práci se budeme zabývat řešením rozsáhlých soustav lineárních algebraických rovnic, které vznikají diskretizací 3D okrajových úloh pružnosti metodou konečných prvků. Numerická řešení lineárních soustav jsou založena na algoritmu iterační metody sdružených gradientů s předpodmíněním. Při řešení je kladen důraz na využití paralelních výpočetních systémů, proto k hlavním cílům práce patří hledání efektivních paralelních algoritmů pro řešení uvedeného typu úloh, programová implementace odpovídajících řešičů a studium chování programů na modelových úlohách.

Paralelizace metody sdružených gradientů je založena na metodách rozložení prostoru, které dovolují dekompozici datových struktur úlohy. Jedná se o separaci složek posunutí a rozklad oblasti na překrývající se podoblasti. Metody rozkladu prostoru umožňují zároveň konstruovat efektivní předpomiňovače využívající řešení na podprostorech, jenž korespondují s podúlohami.

Kromě paralelních algoritmů v práci rovněž ukážeme dvě úpravy algoritmu metody sdružených gradientů. První, dodatečná ortogonalizace směrů postupu při hledání řešení, umožňuje metodě pracovat s obecným typem předpodmiňovače. Druhá, projekce pro řešení úloh se singulární maticí soustavy, dovoluje metodě nalézt zobecněné řešení lineárního systému.

Součástí práce je stručný popis realizovaných řešičů, výsledky testování jejich spustitelných kódů na modelových úlohách z geomechaniky a srovnání ekonomiky řešení na rozdílných platformách paralelních počítačů.

Abstract

The presented thesis deals with large-scale linear systems arising from the finite element discretization of elasticity problems in 3D. To get numerical solutions of these linear systems, we shall use iterative algorithms based on the preconditioned conjugate gradient method and implemented on nowaday powerful parallel computer systems. The main aims of the thesis are the searching for effective algorithms, primarily in parallel, implementations of appropriate solvers and a study of their behaviour on benchmark problems.

The parallelization of the conjugate gradient method will be based on the two different space decomposition techniques enabling to split data structures of the solved problem: the displacement decomposition and the domain decomposition. These methods allow also to construct efficient preconditioners by using the solution of subproblems corresponding to subspaces.

Except the parallel algorithms, we shall describe two modifications of the conjugate gradient method. The first one will be the additional orthogonalization of

search directions, which enables the conjugate gradient method to work with a general preconditioner. The second one will use a projection for solving the singular linear systems.

A part of the thesis will be denoted to a short description of the realized solvers, results of testing their executable codes on benchmark problems in geomechanics and a comparison of a solution on different platforms of parallel computers.

1 Úvod

Řadu inženýrských problémů z oblasti geomechaniky lze matematicky modelovat prostřednictvím 3D úloh pružnosti. Příkladem může být modelování vývoje napěťových polí podzemního dolu podle předpokládaného postupu těžby. K řešení takových úloh se často používá analýza metodou konečných prvků, jenž vede na problém řešení rozsáhlé soustavy lineárních algebraických rovnic. Velikost vzniklé soustavy, běžně dosahující miliónů rovnic, určuje značnou obtížnost nalezení jejího řešení.

Úlohy tohoto typu se obvykle řeší použitím specializovaných řešičů, jejichž základem bývá některá z iteračních metod, které mají v těchto případech příznivější vlastnosti než metody přímé. Prakticky se používá metoda sdružených gradientů, jenž hledá optimální aproximované řešení na Krylovově prostoru, nevyžaduje žádné parametry, je paměťově nenáročná a její efektivitu lze zvýšit předpodmíněním.

I přes použití specializovaných řešičů však zůstává výpočetní čas počítače příliš dlouhý a hledají se techniky, jak jej zkrátit. Jedním z možných přístupů je rozdělení celkové výpočetní práce mezi skupinu spolupracujících procesorů tak, aby velikost řešeného problému mohla být větší a přitom byl výpočet dokončený dříve. Tato myšlenka je zároveň základním principem paralelizace, jenž vede výrobce počítačů ke konstrukci paralelních výpočetních systémů a vývojáře potom k tvorbě paralelních aplikací.

1.1 Vývoj a současné poznatky

Uvedené myšlenky naznačují velkou důležitost problematiky, z níž se odvíjí stále větší zájem a velké množství nových poznatků v oblasti řešení lineárních soustav. Přestože na počátku 50. let byla v práci [1, Hestenes, Stiefel] poprvé představena metoda sdružených gradientů jako samostatná iterační metoda, k jejímu rozvoji došlo až v 70. letech, například díky práci [2, Reid].

Další etapa rozvoje souvisí s předpodmíněním. Začátkem 70. let se objevilo předpodmínění neúplnou faktorizací a v práci [4, Meijerink, van der Vorst] bylo ukázáno, že takové předpodmínění lze konstruovat, pokud je matice soustavy M-maticí. Pro řešení lineárních systémů vzniklých diskretizací úloh pružnosti bylo nutné kombinovat neúplnou faktorizaci s technikou separace složek posunutí. Tuto techniku popisují práce [5, Axelsson, Gustafsson] a [12, Blaheta].

Od poloviny 80. let se dostávají do popředí zájmů také metody založené na rozkladu oblasti, jenž byly prezentovány v pracích mnoha autorů, například ve sbornících [8, Glowinski a kol.], [9, Chan a kol.]. Druhý sborník dokládá, že se v té době současně objevují první práce týkající se využití paralelních počítačů. Jedná se především o možnost aplikace uvedených metod na vektorových počítačích a symetrických multiprocesorech. Během posledního desetiletí potom přibývají práce, které se zabývají návrhy a implementacemi původních paralelních

algoritmů pro různé paralelní výpočetní systémy, například knihy [18, Smith, Bjorstad, Gropp], [23, Dongarra a kol.] a další.

1.2 Hlavní cíle disertační práce

Hlavní cíle předložené práce jsou:

- Návrhy efektivních algoritmů, založených na metodě sdružených gradientů s předpodmíněním a určených pro řešení 3D okrajových úloh pružnosti.
- Paralelizace algoritmů a konstrukce předpodmiňovačů s využitím metod rozkladu prostoru.
- Programová implementace algoritmů se záměrem vytvořit určitou třídu řešičů, na úrovni zdrojových kódů přenositelných mezi různými platformami moderních paralelních výpočetních systémů a navazujících na programový systém `GEM` určený pro matematické modelování reálných problémů z oblasti geomechaniky.
- Testování řešičů na vybraných modelových úlohách, studium jejich chování a ověření jejich funkčních možností. Využití řešičů pro potřeby řešení úloh z praxe.

1.3 Stručný přehled disertační práce

Vlastní práce začíná následující druhou kapitolou, ve které se obecně seznámíme s technickými prostředky a nástroji pro vytváření a provozování paralelních programů, s technikami návrhu paralelních algoritmů a s možnostmi detailní analýzy výsledných spustitelných paralelních kódů včetně posouzení jejich efektivity. V oddíle 2.1.6 jsou charakteristiky konkrétních paralelních počítačů využitých v práci. Zvláštní význam pro programové implementace řešičů uvedených v dalších kapitolách má část oddílu 2.4.1 popisující návrh univerzálního rozhraní pro vzájemné interakce paralelních procesů.

Třetí kapitola nás uvede do oblasti řešení úloh pružnosti. Nejprve se v oddílech 3.1 a 3.2 stručně seznámíme s tímto typem úloh i s praktickými postupy jejich řešení, potom bude v oddíle 3.3 následovat popis modulárního programového systému `GEM`. Vybrané příklady modelových úloh využité při testování řešičů uvedeme v oddíle 3.4.

Stěžejní část práce představují algoritmy založené na metodě sdružených gradientů s předpodmíněním, kterými se budeme zabývat ve čtvrté kapitole. V oddíle 4.1 popíšeme klasický sekvenční algoritmus uvedené iterační metody, různé typy předpodmínění, algoritmus zobecněné metody a projekci dovolující metodě řešit singulární úlohy. Dále v oddíle 4.2 uvedeme metody rozložení prostoru, zejména separaci složek posunutí a rozklad oblasti s překrytím. Tyto metody umožňují

konstrukci jednoúrovňových a dvojúrovňových předpodmiňovačů a zároveň také paralelizaci metody sdružených gradientů, jenž popíšeme v oddíle 4.3.

V páté kapitole se ve stručnosti seznámíme s realizovanými sekvenčními a paralelními programy, s popisem jejich charakteristických vlastností i funkčních možností. Rovněž naznačíme hlavní směry úprav, které zapříčinily postupný vývoj a optimalizaci kódů řešičů.

Numerické experimenty provedené na modelových úlohách budou popsány v šesté kapitole. Testování spustitelných kódů sekvenčních a zejména paralelních řešičů mělo zásadní význam pro rozbor jejich chování, odzkoušení jejich nových funkčních možností a posouzení jejich reálné efektivity. Výsledky testů jsou rovněž důležité z hlediska srovnání ekonomiky řešení (nejen) okrajových úloh pružnosti na různých platformách paralelních počítačů s odlišnými pořizovacími a provozními náklady.

2 Paralelní výpočty

Od samého počátku vývoje počítačů vznikla potřeba jejich větší výpočetní síly. Proto přední světové firmy neustále hledají nové způsoby, kterými by bylo možné zvýšit výkon počítače. Přímočarou cestou může být spřažení více procesorů do jednoho celku, což vede ke konstrukci paralelních počítačů, ale zároveň vyžaduje také vytvoření vhodných paralelních algoritmů a jejich programovou implementaci. Cílem těchto snah je rozdělit výpočetní zátěž mezi spolupracující procesory tak, aby počítaná úloha mohla být větší a byla vyřešena dříve.

2.1 Počítačové architektury

Vlastnosti konkrétního počítače, například vhodnost jeho použití pro požadovanou činnost, možnosti jeho efektivního programování, způsoby jeho rozšiřování nebo zvyšování výkonu a další, se odvíjejí od typu architektury počítače. V následujících oddílech popíšeme běžné typy architektur a budeme se zabývat možnými způsoby implementace paralelních algoritmů.

2.1.1 Flynnova klasifikace počítačů

Nejstarší klasifikaci počítačů vzhledem k jejich schopnostem vykonávat paralelní výpočty provedl Flynn. Ve své práci [3] počítače klasifikuje podle způsobu zpracování toků instrukcí a dat na:

SISD *(Single Instruction, Single Data stream)*: Jeden tok instrukcí vykonávaných jedním procesorem zpracovává jeden tok dat. Tato klasická architektura von Neumannova typu je základem sekvenčních počítačů.

SIMD *(Single Instruction, Multiple Data stream)*: Každá instrukce se v jednom okamžiku vykonává na všech procesorech v počítači, přitom ale každý procesor zpracovává svá jedinečná data. Tato architektura představuje vektorové, případně maticové počítače.

MIMD *(Multiple Instruction, Multiple Data stream)*: Několik různých toků instrukcí je vykonávaných jednotlivými procesory, přitom každý procesor zpracovává svůj tok dat. Tato architektura charakterizuje symetrické multiprocesory, multipočítače a distribuované výpočetní systémy.

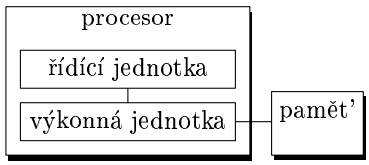
Časem se jednotlivé kategorie této klasifikace rozšířily a vznikaly kategorie nové, které však můžeme s menší či větší přesností zařadit do původníklasifikace.

Poznamenejme, že řazení paralelních počítačů do jednotlivých kategorií není zcela jednotné. Je to způsobeno možnými rozdílnými hledisky posuzování, určitým překrytím významů některých pojmů, například procesor - počítač, a v neposlední řadě i rychlým vývojem v oblasti informatiky obecně.

2.1.2 Sekvenční počítače

Základní architektura počítačů vychází z von Neumannova sekvenčního modelu, který je schématicky znázorněn na obrázku č. 1. Procesor obsahuje řídící jednotku a výkonnou jednotku. Řídící jednotka jednotlivé instrukce dekóduje a připravuje pro výkonnou jednotku, která je provádí. Při zpracování instrukcí výkonná jednotka přistupuje k datům uloženým v operační paměti. Pro urychlení takových přístupů se obvykle mezi procesor a operační paměť řadí rychlá vyrovnávací paměť *(cache memory)*. Celý počítač je doplněn dalšími periferiemi, například vstupně/výstupními jednotkami.

Zvýšení výpočetního výkonu sekvenčního počítače je možné docílit vyšší hustotou integrace součástek a využitím technologií umožňujících zvýšit rychlost operací. Tyto možnosti nepříznivě ovlivňuje konečná rychlost šíření elektrického signálu a především nutnost odvádět značné tepelné ztráty z malého objemu integrovaných obvodů.

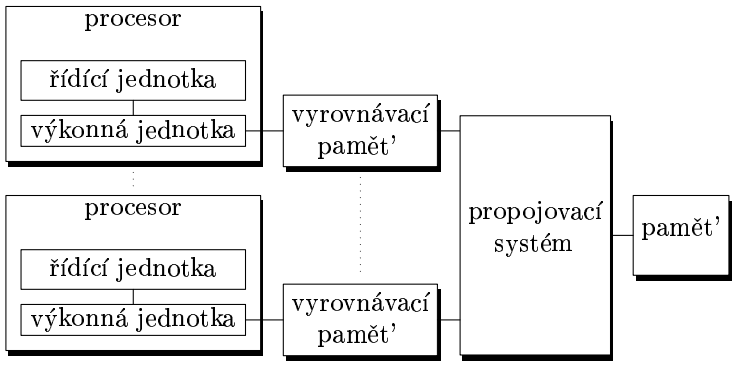


Obrázek 1: Model počítače von Neumannova typu.

V dalším textu si všimneme některých komerčně nejúspěšnějších paralelních architektur. Úmyslně opomineme vektorové počítače a zdůrazníme vlastnosti počítačů s architekturou MIMD, pro které navrhneme paralelní algoritmy, realizujeme odpovídají programy a provedeme testy na modelových úlohách.

2.1.3 Symetrické multiprocesory

Architektura symetrických multiprocesorů je zachycena na obrázku č. 2. Počítač obsahuje sadu navzájem ekvivalentních procesorů, které mají stejné možnosti využití systémových zdrojů, především sdílené operační paměti, a z hlediska operačního systému jsou rovnocenné.



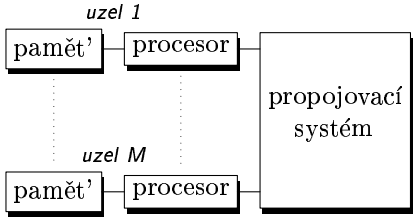
Obrázek 2: Symetrický multiprocesor.

Všechny procesory mohou vykonávat stejný programový kód, jádro operačního systému nemusí při většině svých akcí rozlišovat jednotlivé procesory a může

s nimi zacházet anonymně. Sdílená paměť umožňuje lehce a efektivně implementovat všechny formy interakce paralelních procesů, které budou dále popsány v oddíle 2.2.2.

Kritickým prvkem symetrických multiprocesorů je propojovací systém, na jehož propustnosti silně závisí počet procesorů, které lze do počítače zapojit. Otázky různých konstrukcí propojovacích systémů a jejich charakteristických vlastností budou diskutovány v oddíle 2.1.5. Poznamenejme, že u komerčně úspěšných počítačů tohoto typu nepřekračuje počet procesorů obvykle tři desítky. Důvodem je vysoká náročnost výroby kvalitního propojovacího systému, od níž se potom odvíjí cena celého počítače.

2.1.4 Masivně paralelní systémy



Obrázek 3: Schéma multipočítače.

Na obrázku č. 3 je schéma masivně paralelního systému, jenž bývá také někdy označován jako multipočítač, případně multiprocesor s distribuovanou pamětí. Jde o volně vázanou víceprocesorovou architekturu, jejímiž základními stavebními prvky jsou procesory, které jsou mezi sebou vázané komunikačními linkami. Každý procesor je doplněn o svou lokální operační paměť, v níž má zapsán vlastní programový kód. Procesory proto nemusí být nutně ekvivalentní. Potom ovšem pro každý typ procesoru musí být vždy zvlášť přeložena alespoň určitá část paralelního programu, která odpovídá procesům určeným pro daný typ procesoru.

Celý multipočítač nemusí fyzicky nutně tvořit jeden celek. Jednotlivé výpočetní prvky, které nazveme uzly, mohou být tvořeny samostatnými počítači nejen s rozdílnými procesory, ale v extrémním případě i s rozdílnými operačními systémy, s rozdílnou vnitřní reprezentací dat a podobně. Funkce propojovacího systému potom přebírá počítačová síť, obecně i velmi rozsáhlá a heterogenní, jejíž jednotlivé segmenty mohou být propojeny různými technologiemi. Takovému výpočetnímu systému říkáme klastr.

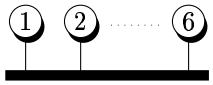
Procesory multipočítače vzájemně komunikují předáváním zpráv. Sdílení dat není možné implementovat, neboť v multipočítači není přítomna žádná sdílená paměť adresovatelná přímo strojovou instrukcí procesorů. Obě formy komunikace procesorů budou popsány dále v oddíle 2.2.2.

Ze schématu architektury multipočítačů je patrné, že její největší výhodou je snadná rozšiřitelnost o další výpočetní prvky, neboť maximální počet procesorů v systému není principiálně omezen a běžně dosahuje stovek procesorů. Naopak evidentními nevýhodami jsou pomalá komunikace mezi jednotlivými procesory stejně jako přístup procesoru k datům, které nemá uloženy ve své lokální operační paměti. Navíc konkrétní topologie propojovacích systémů jsou vhodné pouze pro určité, omezené spektrum aplikací.

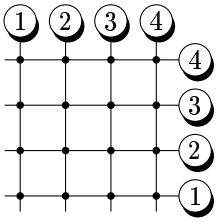
V případě heterogenních počítačových klastrů mohou rozdílné výkony jednotlivých uzlů nepříznivě ovlivnit efektivitu paralelních programů. Proto by součástí každé implementace paralelního algoritmu měl být i algoritmus vyvažování zátěže jednotlivých procesorů. Potřeba a důležitost takových algoritmů roste zejména v poslední době, kdy se stále častěji objevují kombinované klastry, jejichž výpočetní uzly tvoří symetrické multiprocesory.

2.1.5 Propojovací systémy

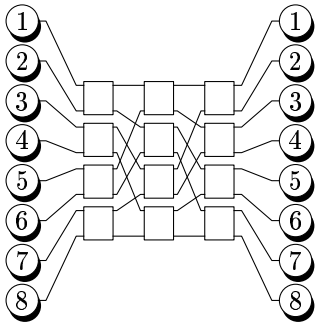
Prostřednictvím propojovacího systému každý procesor paralelního počítače pracuje s daty v operační paměti a komunikuje s ostatními procesory. K důležitým vlastnostem, které vyplývají z konstrukce propojovacího systému a charakterizují jeho kvalitu, patří zejména propustnost dat, náročnost výroby a rozšiřitelnost počtu připojených zařízení. Kvalita propojovacího systému silně ovlivňují pořizovací cenu celého počítače a zároveň klade omezení na maximální počet procesorů počítače. Běžně používané typy propojovacích systémů jsou například sběrnice, křížový přepínač a Ω-síť, méně běžné potom hyperkrychle, mřížka, torus a další.



Obrázek 4: Sběrnice.



Obrázek 5: Křížový přepínač.



Obrázek 6: Ω–síť.

Sběrnice: Na obrázku č. 4 je schématicky znázorněna sběrnice v konfiguraci se šesti připojenými zařízeními. Je to nejlevnější, ale zároveň také nejpomalejší typ propojovacího systému. Pokud některý z procesorů sběrnici využívá, musí ostatní procesory na komunikaci nebo přístup k datům čekat až do okamžiku jejího uvolnění. Aby se nepříznivé vlastnosti sběrnice alespoň částečně eliminovaly, instaluje se mezi ni a procesory vyrovnávací paměť. Tím se však propojení komplikuje a prodražduje. Největší výhodou sběrnice je snadná rozšiřitelnost počtu zařízení, které k ní mohou být připojeny. Proto se používá například v symetrických multiprocesorech, které nemají blokově rozdělenou operační paměť, v pracovních stanicích a osobních počítačích nebo při jejich vzájemném propojení do sítě či klastru technologií Ethernet, Fast Ethernet, ATM a dalšími.

Křížový přepínač: Nejrychlejším, ale také nejdražším typem propojovacího systému je křížový přepínač. Je tvořen N^2 spoji, kde N je maximální počet připojitelných zařízení. V modelovém příkladě na obrázku č. 5 odpovídá

čtyřem připojeným zařízením 16 spojů. Každý procesor je propojen se všemi ostatními a má přímý, bezkolizní přístup do operační paměti. Tyto vlastnosti jsou největšími výhodami křížového přepínače. K nevýhodám patří náročnost technické realizace, s tím spojené vysoké výrobní náklady a technické vlastnosti omezující maximální počet procesorů v systému na méně než 30. Křížový přepínač se nejčastěji používá u vektorových počítačů nebo u symetrických multiprocesorů s blokově rozdělenou operační pamětí. Příkladem může být SFI (*Sun Fireplane Interconnect*) použitý v každém stroji Sunfire 6800 výpočetního klastru SUN-2.

Ω–síť: Z hlediska rychlosti a pořizovací ceny je Ω–síť kompromisem mezi sběrnicí a křížovým přepínačem. Je realizována několika stupni přepínačů, které propojují každé připojené zařízení se všemi ostatními. Počet stupňů Ω–sítě je závislý na typu použitých přepínačů, zejména na počtu jejich vstupů a výstupů, a také na počtu propojovaných zařízení, který není principiálně omezen. Modelový příklad na obrázku č. 6 znázorňuje vzájemné propojení osmi zařízení pomocí přepínačů se dvěma vstupy a dvěma výstupy. Stejně jako u sběrnice, jsou u Ω–sítě možné kolize, které snižují rychlost propojení. Běžně se tento typ propojovacího systému používá v masivně paralelních počítačích, jeho zástupcem může být HPS (*High Performance Switch*) v IBM SP.

2.1.6 Příklady paralelních počítačů

Charakteristiky počítačů využitých pro disertační práci:

IBM SP (VŠB–TU Ostrava): IBM SP masivně paralelní systém, využity 4 uzly (Power2/67) s pamětí 128 MB, propojovací systém ATM, operační systém AIX 4, systém předávání zpráv MPI (MPICH) a PVM.

SUN-1 (EPCC Edinburgh): SUN Enterprise HPC 6500, symetrický multiprocesor, 18 procesorů (UltraSparc-II/400), sdílená paměť 18 GB, operační systém Solaris, systém předávání zpráv MPI (SUN).

LINUX-1 (EPCC Edinburgh): PC klastr, server + 16 uzlů (AMD Athlon/650), každý s pamětí 128 MB a 2× Fast Ethernet rozhraním, operační systém Linux (RedHat), systémy předávání zpráv MPI (MPICH) a PVM.

LINUX-2 (ÚGN AV Ostrava): PC klastr, server (Intel Pentium III/450) s pamětí 128 MB + 3 uzly (Intel Pentium/133) s pamětí 32 MB, (Fast) Ethernet rozhraní, operační systém Linux (RedHat), systém předávání zpráv PVM.

LINUX-3 (VŠB–TU Ostrava): PC klastr, 4 uzly (Intel Pentium III/500) s pamětí 384 MB, Fast Ethernet rozhraní, operační systém Linux (Debian), systém předávání zpráv PVM.

LINUX-4 (ÚGN AV Ostrava): PC klastr, 8 uzlů (AMD Athlon/1400) s pamětí 768 MB, 2× Fast Ethernet rozhraní, operační systém Linux (Debian), systémy předávání zpráv MPI (MPICH 1.2.1 a LAM 6.5.6) a PVM 3.4.2, knihovna PETSc 2.1.3.

SUN-2 (EPCC Edinburgh): SUN HPC klastr, 2× symetrický multiprocesor Sunfire 6800, každý 24 procesorů (UltraSparc-III/750) a sdílená paměť 48 GB, operační systém Solaris 2.8 se SUN Grid Engine, systém předávání zpráv MPI (SUN), knihovna PETSc 2.1.3.

2.2 Paralelizace algoritmů

Vlastnímu programování paralelní aplikace zpravidla předchází fáze analýzy algoritmu, která obvykle sestává z několika kroků: dekompozice výpočtů a příslušných dat, řešení komunikace, aglomerace elementárních činností a mapování paralelních procesů na příslušné procesory. Poznamenejme, že otázky návrhu a konstrukce paralelních algoritmů jsou diskutovány například v [15].

2.2.1 Dekompozice výpočtů a dat

Dekompozice určuje základní přístup k paralelizaci algoritmu. Jedná se o rozložení činnosti na jednotlivé složky (procesy), které je možné vykonávat současně. Rozlišujeme dva základní modely dekompozice:

Funkční: Celý výpočet je rozdělen na samostatné činnosti, které mohou být vykonávány paralelně a pracují s malým objemem dat. Tento přístup využijeme pro relativně složitou činnost, kterou rozdělíme na menší části, k nimž logicky přiřadíme příslušná data. Přitom je proti sekvenčímu algoritmu výhodou přehlednější strukturování jednotlivých paralelních procesů a možnost efektivně využít i paralelní počítače s různými typy procesorů. Příkladem aplikace mohou být simulační a monitorovací programy nebo řízení technologických procesů v reálném čase.

Datová: Při paralelizaci algoritmu jsou nejprve logicky rozdělena objemná data, kterým se následně přiřadí jednotlivé procesy, jenž je zpracovávají. Obvykle všechny procesy vykonávají stejnou činnost, ale každý nad jinou částí dat. Tím je sledováno především výkonnostní hledisko s cílem maximálně urychlit výpočet. Takový přístup využijeme například při paralelním násobení matic, při paralelním skalárním součinu vektorů a při mnoha dalších operacích lineární algebry. Proto datovou dekompozici využívá většina paralelních algoritmů.

V praxi se můžeme setkat i s dekompozicí hybridní, která je kombinací obou základních modelů.

Při dekompozici nemusíme zohledňovat konkrétní typ dostupné počítačové architektury, neboť uváděné modely dekompozice jsou obecné a lze je proto realizovat na paralelních počítačích se sdílenou i distribuovanou pamětí.

2.2.2 Řešení komunikace

Komunikace a řešení otázek iterakce procesů je druhým krokem analýzy paralelního algoritmu. Po dekompozici nejsou paralelní procesy obecně nezávislé, je nutné řešit výměnu dat. Při návrhu komunikačních toků bychom se měli snažit o realizaci zejména následujících vlastností paralelního programu:

- minimální interakce a co největší samostatnost paralelních procesů, neboť častá komunikace zpomaluje celý výpočet, zvláště u masivně paralelních počítačů a klastrů;
- škálovatelnost neboli nezávislost na počtu dostupných procesorů.

Při rozhodování, který model komunikace v paralelní aplikaci zvolíme, musíme uvažovat architekturu paralelního počítače a programové prostředky poskytované jeho operačním systémem. Základní modely interakce procesů se liší způsobem výměny informací mezi spolupracujícími procesy:

Sdílení dat: Pro vzájemnou výměnu informací využívají procesy sdílených datových struktur, které jsou uloženy v části společné operační paměti. Proto se technika sdílení dat realizuje výhradně na paralelních počítačích se sdílenou pamětí. Problematická může být současná změna sdílených dat více procesy, kterou je nutné vyloučit. Takové části programu nazýváme kritické sekce a k zabezpečení jejich bezkonfliktního provedení se využívají speciální nízkoúrovňové programové konstrukce, například semafory nebo paměťové zámky.

Předávání zpráv: Výměna informací mezi dvěma komunikujícími procesy probíhá tak, že první z nich - odesílatel - uloží příslušná data obsažená ve zprávě do vyrovnávací paměti, odkud si je následně vyzvedne druhý proces - příjemce. Uvedená technika je jediná možná na paralelních počítačích s distribuovanou pamětí, lze ji však realizovat i na systémech se sdílenou pamětí tím, že se zasílání zpráv emuluje přes sdílenou paměť. Jedná se tedy o univerzální techniku interakce procesů na paralelních architekturách typu MIMD, proto ji upřednostňujeme v případech, kdy požadujeme přenositelnost paralelní aplikace.

Obě uvedené techniky umožňují také vzájemnou synchronizaci paralelních procesů. V prvním případě mohou procesy testovat změnu obsahu sdílených dat, v případě druhém si mohou vyměnit zprávy s nulovým informačním obsahem.

2.2.3 Aglomerace a mapování

Aglomerace a mapování jsou závěrečné kroky analýzy paralelního algoritmu. Při implementaci je důležité vyvážit zátěž jednotlivých procesorů s ohledem na jejich výkon tak, aby efektivita výsledného kódu byla maximální. Proto provádíme revizi dekompozice a komunikace, kterou označujeme jako aglomerace. Cílem aglomerace je sloučení některých elementárních nebo neparalelizovatelných procesů a případně snížení počtu komunikací na úkor replikace výpočtů nebo dat. Procesorům paralelního výpočetního systému jsou následně k vykonání přiděleny výsledné paralelní procesy. Přiřazování procesů jednotlivým procesorům nazýváme mapování a provádí jej uživatel nebo probíhá automaticky. První případ je nejvýhodnější v aplikacích s menším počtem paralelních procesů, které jsou založeny na funkční dekompozici a provozovány na heterogenních systémech s různými typy procesorů. Ve druhém případě automatické mapování programově zajišťuje sama aplikace, ve které je implementován algoritmus pro vyvažování zátěže procesorů[1]. Jeho činnost může být postavena na pravěpodobnostní metodě, metodě typu farmář/dělníci nebo některé jiné. Automatické mapování se používá především v aplikacích s velkým počtem paralelních procesů, které jsou založeny na datové dekompozici.

2.3 Analýza programů

V oddíle 2.2 jsme se zabývali analýzou algoritmů, která zároveň řeší základní otázky praktického postupu při jejich paralelizaci. V dalším textu se soustředíme na analýzu výsledných paralelních programů, jenž je důležitá pro jejich další vývoj. Podrobnosti, jenž se týkají analýzy paralelních programů a hodnocení jejich výkonnosti, se lze dočíst například v [22].

2.3.1 Hodnocení výkonnosti

Od dobrého paralelního programu můžeme očekávat zvýšení výkonu oproti jeho sekvenční verzi, nižší provozní náklady, zrychlení doby provádění a další kvality. V mnoha případech požadavky na jejich splnění stojí proti sobě, pouze výjimečně dosáhneme u aplikace ideálního skloubení uvedených vlastností.

Při posuzování efektivity paralelního programu pohlížíme na celkovou dobu jeho provádění, využití jednotlivých procesorů a jejich vzájemnou komunikaci, paměťové nároky a vstupně/výstupní požadavky programu. Kromě toho můžeme zohlednit také jiné vlastnosti, například přenositelnost programu nebo jeho škálovatelnost. K základním charakteristikám paralelního programu patří zrychlení, účinnost, složitost a cena.

[1] Na homogenních systémech bývají tyto algoritmy velmi jednoduché, jejich význam a složitost však narůstá na systémech heterogenních, které se přirozeně a stále častěji objevují například v souvislosti s postupným budováním a rozvíjením počítačového klastru.

Zrychlení S *(speed up)* můžeme vypočítat ze vztahu

$$S = \frac{T_0}{T_N} ,$$

kde T_0 je doba provádění nejlepšího možného sekvenčního programu, který vykonává stejný výpočet jako paralelní program, jehož celková doba provádění je T_N při použití N procesorů téhož počítače.

Poznamenejme, že T_1 označujeme dobu provádění paralelního programu na jednom procesoru paralelního počítače za předpokladu, že paralelní program běh na jednom procesoru umožňuje. V takovém případě lze charakterizovat paralelní program pomocí relativního zrychlení S_R definovaného vztahem

$$S_R = \frac{T_1}{T_N} .$$

Účinnost E *(efficiency)* neboli míru využití jednotlivých procesorů můžeme jednoduše vypočítat pomocí zrychlení ze vztahu

$$E = \frac{S}{N} .$$

Paralelní program je lepší, pokud $E \to 1$ (neuvažujeme anomální zrychlení).

Složitost $C = C(n)$ *(complexity)* je funkce, jejíž hodnota je pro paralelní program úměrná maximální době jeho provádění pro všechny možné vstupy o rozměru n. Příkladem může být velikost matice soustavy lineárních rovnic v případě iteračních metod, které danou soustavu řeší.

Cena W *(work, cost)* je množství práce úměrné celkovému strojovému času všech použitých procesorů. Můžeme ji vypočítat ze vztahu

$$W = CN .$$

Uvažujeme-li $C = T_N$, potom

$$W = T_N N = \frac{T_0}{E} .$$

Na základě této charakteristiky můžeme vyvíjet cenově optimální algoritmy tak, aby množství vynaložené práce bylo úměrné době provádění nejlepšího známého sekvenčního programu.

2.3.2 Amdahlův zákon

Maximální hodnotu zrychlení paralelního programu vůči sekvenčnímu omezuje Amdahlův zákon. Vychází z předpokladu, že výpočet nelze paralelizovat úplně, určitá jeho část musí být provedena sekvenčně. Tuto část označíme f, zbývající paralelizovatelná část bude $(1 - f)$. Maximální zrychlení S výpočtu vykonaného N procesory bude limitováno

$$S \leq \frac{1}{f + \frac{1-f}{N}} \leq \frac{1}{f} \ .$$

Je nutné si však uvědomit, že k problémům paralelních výpočtů nelze přistupovat na úrovni existujících programů, ale na úrovni algoritmů. Časový podíl sekvenčních výpočtů je možné návrhem vhodného algoritmu minimalizovat, případně zcela vyloučit. Část výpočtů, která je paralelizovatelná, můžeme v ideálním případě zrychlit až N-krát. Dosažitelné zrychlení paralelního programu oproti sekvenčnímu je tedy omezeno hodnotou

$$S \leq N * (1 - f) + f \leq N \ .$$

V praxi může být toto teoretické omezení maximální hodnoty zrychlení překonáno, potom hovoříme o superlineárním nebo též anomálním zrychlení. Důvodem tohoto jevu mohou být například nevhodná konstrukce sekvenčního programu, vedlejší efekty při práci s pamětí nebo použití různých datových struktur v sekvenčním a v paralelním programu.

2.3.3 Rozdělení výpočetního času

U složitějších architektur je nutné sledovat celkovou dobu provádění paralelního programu podrobněji. Jedná se o časový interval mezi spuštěním první a ukončením poslední jeho části. Každá část paralelního programu běží na samostatném procesoru a celkovou dobu jejího běhu T na daném procesoru p můžeme vyjádřit vztahem

$$T = T(p) = T_C + T_M + T_I \ .$$

Výpočetní čas T_C je celková doba provádění výpočetních operací procesorem. Obvykle však nelze uvažovat jednotlivé procesory odděleně, činnost procesoru může být ovlivněna zřetězováním instrukcí, způsobem přístupu k datům uloženým v operační paměti, využitím vyrovnávací paměti. Nelze tedy vyloučit ani závislost rychlosti výpočtu jednoho procesoru na počtu procesorů instalovaných v počítači.

Komunikační čas T_M je celkový čas potřebný pro komunikaci procesů spuštěných na daném procesoru s ostatními procesy. Tato komunikace může proběhnout buď interprocesorově nebo intraprocesorově. Podíl komunikačního času

T_M vůči celkové době provádění T může být velmi významný například u masivně paralelních počítačů, jejichž procesory jsou vzájemně propojeny relativně pomalejšími linkami (vzhledem k mnohem rychlejšímu spojení procesor–paměť).

Časové prodlevy T_I jsou celkový čas, po který byl procesor v klidu, neboť čekal na výpočet, na dočasně nedostupná data nebo na komunikaci. Využití procesoru roste se snižováním podílu časových prodlev T_I na celkové době provádění T. Proto je výhodné, když $T_I \to 0$.

2.4 Programová podpora

Aby mohl aplikační programátor využít schopností paralelního počítače a přitom nebyl závislý na jeho konkrétním typu, potřebuje nástroje, které by mu poskytovaly prostředky pro podporu paralelních programů. K takovým nástrojům patří paralelizující překladače, prostředky pro analýzu a ladění paralelních programů a knihovny uživatelských funkcí, které mohou být volány z aplikace a které zajistí požadované činnosti na konkrétním paralelním počítači. Jedná se například o funkce, jenž realizují předávání zpráv, nebo o procedury pro numerická řešení dílčích podúloh řešeného problému.

2.4.1 Systémy pro předávání zpráv

Knihovny uživatelských funkcí pro předávání zpráv jsou obvykle částí většího celku, který má svoji specifickou koncepci a který označujeme jako systém pro předávání zpráv. Každý takový systém definuje vlastní rozhraní pro předávání zpráv. Ve stručnosti se seznámíme pouze se dvěma nejrozšířenějšími systémy, PVM a MPI, a popíšeme jejich rozhraní. Dále pomocí uživatelských funkcí pro předávání zpráv navrhneme rozhraní univerzální, jenž by mohl využít aplikační programátor požadující maximální přenositelnost paralelního programu. Univerzální rozhraní bude pracovat s prostředky poskytovanými buď PVM nebo MPI a to podle jejich dostupnosti na konkrétním paralelním počítači.

Prostředí PVM

PVM *(Parallel Virtual Machine)* je balík programů a knihoven uživatelských funkcí, který umožnuje vytvořit ze sítě sekvenčních a paralelních počítačů jeden velký paralelní virtuální stroj s architekturou distribuovaného systému. Podrobné informace o PVM verze 3.3 nalezneme v [13].

Jednotlivé hostitelské počítače virtuálního stroje mohou mít odlišné architektury, výpočetní možnosti i kapacity a různé mohou také být technologie jejich vzájemného propojení. Na těchto obecně heterogenních hostitelích vytváří PVM jednotné prostředí pro běh paralelních aplikací, ve kterém zajistí nutnou konverzi formátu dat při vzájemné interakci procesů.

PVM se skládá ze dvou částí. První z nich je démon[2] `pvmd3`, který poskytuje procesům paralelní aplikace služby virtuálního stroje. Po spuštění se `pvmd3` replikuje na každém z uživatelsky definovaných hostitelů virtuálního stroje a naváže spojení se všemi svými kopiemi. Tím je virtuální stroj vytvořen a připraven ke spouštění paralelních aplikací. Druhou částí PVM je knihovna uživatelských funkcí, jejichž prostřednictvím procesy paralelní aplikace komunikují se svými lokálními démony `pvmd3` a využívají tak služeb PVM. Knihovna PVM obsahuje především funkce pro spouštění paralelních procesů, řízení jejich běhů a řešení jejich vzájemné komunikace a synchronizace.

Interakce procesů je založena na modelu předávání zpráv. Vzájemně komunikovat mohou dva procesy nebo celá dynamicky vytvořená skupina procesů, přičemž se složení skupiny může v čase měnit a naopak jeden proces může patřit do několika skupin současně. Paralelní aplikace, která pracuje s dynamickými skupinami úloh, musí volat funkce speciální knihovny. Protože démon `pvmd3` tyto služby neposkytuje, spustí se při prvním volání některé funkce z této knihovny speciální démon, který již patřičné činnosti obstará.

Kromě zmíněných skupin funkcí poskytuje PVM funkce pro dynamickou konfiguraci virtuálního stroje. Nastavení virtuálního stroje je možné prostřednictvím konfiguračního souboru při startu PVM, interaktivně z uživatelské konzoly nebo prostřednictvím procesů, které volají příslušné knihovní funkce. Přitom možnosti všech uvedených způsobů jsou srovnatelné.

Zbývající část knihovny PVM tvoří informační funkce pro mapování aktuálního stavu virtuálního stroje, funkce rozšiřující možnosti komunikace procesů[3] a systémové funkce, například pro čtení aktuální hodnoty časovače.

Programátoři mohou psát zdrojové texty paralelních aplikací pro PVM buď v programovacích jazycích C nebo Fortran[4]. Jejich překlad do spustitelného tvaru probíhá podle stejných pravidel jako v případě běžných aplikací.

Na obrázku č. 7 je fragment zdrojového textu paralelního programu pracujícího pod PVM, jenž je doplněn o stručný komentář popisující jednotlivé kroky. Příklad modeluje doménovou dekompozici libovolného problému, kdy pět identických procesů, které obecně nazýváme dělníci, řeší stejný problém, ovšem každý zpracovává svoji vlastní porci dat. Celý výpočet řídí další proces nazývaný farmář, k jehož hlavním funkcím patří rozdělení práce mezi dělníky, shromáždování dílčích výsledků a řešení vzájemné synchronizace všech procesů.

[2] Termín *démon* je znám z operačního systému Unix. Je to obvykle trvale běžící proces, jenž pro ostatní procesy zprostředkovává určité služby.

[3] Interakce procesů je rozšířena o možnost vyslání signálů nebo generování asynchronních zpráv na základě výskytu události.

[4] Všechny příklady zdrojových textů uvedené v této práci jsou napsány v programovacím jazyce Fortran 77.

```
  PROGRAM pvmfarma

C definice konstant:
C     npr ... počet dělníků

  PARAMETER( npr=5 )

C deklarace proměnných:
C     myid ... identifikační číslo procesu
C      ids ... pole identifikačních čísel (všechny procesy ve farmě)
C     nprs ... počet úspěšně spuštěných dělníků
C      msg ... třída zprávy
C     info ... pomocná (informační) proměnná

  INTEGER myid, ids(0:npr), nprs, msg, info

C start procesu: zjistím identifikační číslo sebe i svého rodičovského procesu

  CALL pvmfmytid( myid )
  CALL pvmfparent( ids(0) )

C identifikace procesu: určím, zda-li jsem farmář nebo dělník;
C • pokud neexistuje můj rodičovský proces, jsem farmář

  IF( ids(0) .LT. 0 ) THEN

C   kód farmáře:
C     - spustím dělníky (kopie programu pvmfarma)
C     - pošlu všem dělníkům zprávu třídy 1 obsahující ids(1), ..., ids(npr);
C       tím bude každý proces znát všechny ostatní a může s nimi spolupracovat (komunikovat)
C     - voláním funkce farmar_prace() vykonám veškerou činnost specifikovanou pro farmáře

    CALL pvmfspawn( 'pvmfarma', 0, '*', npr, ids(1), nprs )

    msg = 1
    CALL pvmfinitsend( 0, info )
    CALL pvmfpack( INTEGER4, ids(1), npr, 1, info )
    CALL pvmfmcast( npr, ids(1), msg, info )

    CALL farmar_prace( ... )

C • jelikož jsem potomek, jsem dělník

  ELSE

C   kód každého dělníka:
C     - přijmu zprávu třídy 1 s příslušným obsahem
C     - voláním funkce delnik_prace() vykonám veškerou činnost specifikovanou pro dělníka

    msg = 1
    CALL pvmfrecv( ids(0), msg, info )
    CALL pvmfunpack( INTEGER4, ids(1), npr, 1, info )

    CALL delnik_prace( ... )

  ENDIF

C ukončení procesu: přestávám využívat služby virtuálního stroje; končím

  CALL pvmfexit( info )
  STOP
  END
```

Obrázek 7: Kostra paralelní aplikace pro PVM.

Rozhraní MPI

MPI *(Message Passing Interface)* je specifikace rozhraní u knihoven funkcí pro předávání zpráv a na rozdíl od PVM se nejedná o konkrétní sadu programů. Hlavním účelem této specifikace je vytvoření standardu pro paralelní zpracování zejména na paralelních počítačích s distribuovanou pamětí. Detailně se můžeme se standardem MPI ve verzi 1.1 seznámit prostřednictvím [16].

Poznamenejme, že ke známým programovým implementacím MPI patří například MPICH nebo LAM. Podrobné informace o uvedených volně dostupných softwarových produktech a aktuální verze jejich programových kódů, MPICH verze 1.2.3 a LAM verze 6.5.6, můžeme získat z Internetových adres [27], [28].

K cílům projektu MPI patří především návrh aplikačního programového rozhraní, které by bylo možné efektivně implementovat v heterogenním prostředí tak, aby aplikační programátor nemusel zasahovat do nižších systémových vrstev. Návrh rozhraní vychází z nejrozšířenějších implementací systému předávání zpráv, které dříve byly vlastnická řešení předních firem vyvíjejících paralelní počítače. Z koncepce MPI vyplývá podpora především systémů s distribuovanou pamětí, ovšem běžně se používá i na systémech se sdílenou pamětí.

Standard MPI zahrnuje komunikaci mezi dvěma procesy, kolektivní operace, podporu skupin procesů, komunikační kontexty (též komunikační prostory, komunikační světy) nebo možnosti využití topologií procesů. Naopak nezahrnuje explicitní operace pro systémy se sdílenou pamětí, prostředky pro vývoj a ladění paralelních aplikací nebo vstupně/výstupní funkce. MPI také nedefinuje uživatelské prostředí, neuvádí žádný způsob spouštění paralelních aplikací a nezahrnuje možnosti konfigurace paralelního výpočetního prostředku. To vše určuje až konkrétní implementace MPI.

K největším rozdílům MPI oproti PVM, které bezprostředně souvisí s hlavními principy pro předávání zpráv v uvedených systémech, patří:

- Každý MPI-proces je asociován s alespoň jedním komunikátorem, což je celočíselná hodnota sloužící jako prostředek pro specifikování konkrétního komunikačního kontextu, jenž není explicitním objektem. V rámci jednoho komunikačního kontextu jsou možné například komunikace procesů (i hromadné) nebo provádění některých operací. V PVM existuje pouze jediný, pro všechny procesy společný komunikační prostor.

  MPI definuje dva typy komunikátorů. Intra–komunikátor slouží ke specifikaci komunikačního světa pro paralelní procesy náležející do stejné skupiny, oproti tomu inter–komunikátor vytváří komunikační prostor pro dvě skupiny procesů. Tato separace vnitřní a vnější komunikace procesu je výhodná především z hlediska bezpečnosti a ochrany zpráv.

- V obou systémech jsou rozdílné také možnosti výběru vhodných komunikačních funkcí pro posílání zpráv, neboť MPI, jehož skupina funkcí je mnohem bohatější, definuje například blokující i neblokující odeslání zprávy,

přičemž každá funkce má čtyři varianty v závislosti na zvoleném komunikačním režimu. Samozřejmostí potom jsou funkce pro blokující a neblokující příjem zprávy. Možnosti interakce MPI-procesů dále rozšiřuje široká škála funkcí pro hromadnou komunikaci procesů nebo funkce pro současné odeslání a příjem zpráv, která zabraňuje případným kolizím, jež mohou být příčinou zablokování výpočtu (deadlock).

V souvislosti s komunikací poznamenejme, že MPI nechybí možnost vytvářet za běhu programu odvozené datové typy z již existujících typů. Tyto odvozené typy můžeme použít například při přenosu nespojitých, heterogenních dat.

- MPI obsahuje rovněž mechanismus pojmenování procesů v komunikačním kontextu, kterým se popisuje komunikační schéma. Tento mechanismus, který se nazývá virtuální topologie, se využívá pro optimalizaci komunikace, k efektivnímu mapování procesů a případně i ke zlepšení čitelnosti zdrojových textů programů.

- Ke zefektivnění opakovaných komunikací jednoho druhu, například tatáž komunikace mezi dvěma procesy v cyklu, můžeme využít persistentní komunikační požadavky. MPI nabízí funkce, na jejichž základě dochází k redukci nutné režie opakované komunikace.

Standard MPI definuje rozhraní pro programovací jazyky C a Fortran, programátor má z hlediska komunikace úloh přinejmenším srovnatelné možnosti jako v případě použití PVM. Překlad programů do spustitelného tvaru se neodlišuje od běžně používaných technik.

Přestože počet funkcí MPI je více než dvojnásobný oproti PVM, k realizaci paralelní aplikace založené na modelu posílání zpráv jich stačí pouze šest, což dokládá komentovaný příklad zdrojového textu pro MPI zachycený na obrázku č. 8. Program řeší doménovou dekompozici obecného problému (podobně jako příklad na obrázku č. 7 pro PVM), kdy celý výpočet řídí farmář, jenž rozděluje práci pro obecně libovolný počet dělníků.

```
  PROGRAM mpifarma

C deklarace proměnných:
C     myid ... identifikační číslo procesu
C     nprs ... počet všech spuštěných procesů
C      msg ... třída zprávy
C   status ... pomocná (informační) proměnná
C     info ... pomocná (informační) proměnná
C        i ... pomocná proměnná (pro programovou smyčku)
C     data ... datové pole s 10. reálnými čísly

  INTEGER  myid, nprs, imsg, status, info, i
  REAL     data(10)

C start procesu: zjistím své identifikační číslo a počet spuštěných procesů
C                Pozn.: MPI_COMM_WORLD je předdefinovaný komunikátor pro
C                       specifikaci komunikačního prostoru, do kterého patří
C                       všechny paralelní procesy

  CALL MPI_INIT( info )
  CALL MPI_COMM_RANK( MPI_COMM_WORLD, myid, info )
  CALL MPI_COMM_SIZE( MPI_COMM_WORLD, nprs, info )

C identifikace procesu: určím, zda-li jsem farmář nebo dělník;
C • jsem první spuštěný proces, jsem farmář

  IF( myid .EQ. 0 ) THEN

C   kód farmáře:
C     - pošlu všem dělníkům zprávu třídy 1 obsahující data
C     - voláním funkce farmar_prace() vykonám veškerou činnost specifikovanou pro farmáře

    msg = 1
    DO 10 i = 1,nprs
10     CALL MPI_SEND( data, 10, MPI_REAL, i, msg, MPI_COMM_WORLD, info )

    CALL farmar_prace( ... )

C • nejsem první spuštěný proces, jsem dělník

  ELSE

C   kód každého dělníka:
C     - přijmu od farmáře zprávu třídy 1 obsahující data
C     - voláním funkce delnik_prace() vykonám veškerou činnost specifikovanou pro dělníka

    msg = 1
    CALL MPI_RECV( data, 10, MPI_REAL, 0, msg, MPI_COMM_WORLD, status, info )

    CALL delnik_prace( ... )

  ENDIF

C ukončení procesu: přestávám využívat služby virtuálního stroje; končím

  CALL MPI_FINALIZE( info )
  STOP
  END
```

Obrázek 8: Návrh paralelního programu založeného na MPI.

Univerzální rozhraní

V posledních letech se vývojáři stále častěji setkávají s požadavky na univerzálnost a přenositelnost aplikací, s rozvojem paralelních počítačů potom tyto požadavky postupně začaly vznikat také u paralelních programů. Protože PVM a MPI mají výsadní postavení mezi nástroji pro tvorbu paralelních aplikací na symetrických multiprocesorech i masivně paralelních a distribuovaných systémech, jejichž počet začíná být ve světě paralelních počítačů dominantní, naznačíme možnosti programování univerzálních aplikací, které by bylo možné provozovat buď pod PVM nebo pod některou z implementací MPI.

```
      PROGRAM unifarma

C     definice konstant:
C         maxp maximální počet procesů
C         dlen délka pole, velikost dat
      PARAMETER( maxp=30 )
      PARAMETER( dlen=10 )

C     definice proměnných:
C          mid pořadí procesu
C          ids pole identifikačních čísel
C          num počet spuštěných procesů
C         data demonstarační data
      INTEGER*4 mid, ids(maxp), num, data(dlen)

C     začátek procesu
      CALL mpstart( mid, ids, num )

C     jsem dělník nebo farmář ?
      IF( mid .EQ. 1 ) THEN

C       kód farmáře
        CALL mpmcsti( data, dlen, ids(2), np-1, 3 )
        CALL farmar_prace( ... )

      ELSE

C       kód dělníků
        CALL mprecvi( data, dlen, ids(1), 3 )
        CALL delnik_prace( ... )

      ENDIF

C     ukončení procesu
      CALL mpstop( )

      END
```

Obrázek 9: Kostra univerzální aplikace.

Pro demonstraci se omezíme na vyvíjení paralelní aplikace `unifarma`, která bude provádět stejnou činnost jako aplikace `pvmfarma` a `mpifarma` z příkladů na obrázcích č. 7, 8. Aplikace `unifarma` bude schopna navázat a ukončit komunikaci s patřičnou paralelní knihovnou a její procesy budou moci mezi sebou komunikovat posíláním zpráv.

Nezbytným předpokladem k dosažení uvedeného cíle je důsledné oddělení volání funkcí paralelní knihovny do samostatného modulu, jehož funkce bude naše aplikace využívat přes univerzální rozhraní. Součástí zdrojových textů aplikace budou dva moduly (jeden pro PVM, druhý pro MPI) s funkcemi pro komunikaci s konkrétní paralelní knihovnou. Obrázek č. 9 zachycuje kostru aplikace `unifarma` a obrázek č. 10 návrhy modulů pro komunikaci s paralelními knihovnami. Pro překlad zdrojových textů do spustitelného tvaru zvolíme vždy jeden z modulů podle toho, kterou paralelní knihovnu na daném systému použijeme.

Stejným způsobem, tj. oddělením implementačně závislých funkcí do samostatných modulů, bychom postupovali také v případech vyvíjení přenositelných aplikací, které při svém běhu využívají služeb systémových prostředků specifických pro konkrétní počítačovou architekturu, operační systém a podobně.

```
C  MPSTART: navázání spolupráce s paralelní knihovnou

     SUBROUTINE mpstart( mid, ids, num )

     PARAMETER( nps=6 )

     INTEGER*4   mid, mytid, mypar, ids(*), num, na, i, info

     CALL pvmfmytid( mytid )
     na = iargc()
     IF( na .GT. 0 ) THEN
         mid    = 1
         ids(1) = mytid
         CALL pvmfspawn( 'unifarma', 0, '*', nps, ids(2), info )
         num    = info + 1
         DO 10 i = 2,nps
            IF( ids(i) .GT. 0 ) THEN
              CALL pvmfinitsend( 1, info )
              CALL pvmfpack( 3, num, 1, 1, info )
              CALL pvmfpack( 3, ids, nps, 1, info )
              CALL pvmfsend( ids(i), 0, info )
            ENDIF
10       CONTINUE
       ELSE
          CALL pvmfparent( mypar )
          CALL pvmfrecv( mypar, 0, info )
          CALL pvmfunpack( 3, num, 1, 1, info )
          CALL pvmfunpack( 3, ids, nps, 1, info )
          DO 20 i = 2,nps
             IF( mytid .EQ. ids(i) ) mid = i
20        CONTINUE
     ENDIF
     CALL pvmfsetopt( 1, 1, info )

     END

C  MPRECVI: příjem zprávy obsahující data typu INTEGER*4

     SUBROUTINE mprecvi( buffer, nitem, src, msg )

     INTEGER*4  buffer(*), nitem, src, msg, info

     CALL pvmfrecv( src, msg, info  )
     CALL pvmfunpack( 3, buffer, nitem, 1, info )

     END

C  MPSENDI: odeslání zprávy, data typu INTEGER*4

     SUBROUTINE mpsendi( buffer, nitem, dest, msg )

     INTEGER*4  buffer(*), nitem, dest, msg, info

     CALL pvmfinitsend( 1, info )
     CALL pvmfpack( 3, buffer, nitem, 1, info )
     CALL pvmfsend( dest, msg, info  )

     END

C  MPMCSTI: hromadné odeslání zprávy, data typu INTEGER*4

     SUBROUTINE mpmcsti( buffer, nitem, ids, nids, msg )

     INTEGER*4  buffer(*), nitem, ids(*), msg, nids, info

     CALL pvmfinitsend( 1, info )
     CALL pvmfpack( 3, buffer, nitem, 1, info
     CALL pvmfmcast( nids, ids, msg, info  )

     END

C  MPSTOP: ukončení spolupráce s paralelní knihovnou

     SUBROUTINE mpstop( )

     INTEGER*4  info

     CALL pvmfexit( info )

     END
```

```
C  MPSTART: navázání spolupráce s paralelní knihovnou

     SUBROUTINE mpstart( mid, ids, num )

     IMPLICIT NONE
     INCLUDE  'mpif.h'
     INTEGER  mid, ids(*), num, i, info

     CALL MPI_INIT( info )
     CALL MPI_COMM_RANK( MPI_COMM_WORLD, mid, info )
     CALL MPI_COMM_SIZE( MPI_COMM_WORLD, num, info )

     mid = mid + 1

     DO 10 i = 1,num
       ids(i) = i-1
10   CONTINUE

     END

C  MPRECVI: příjem zprávy, data typu INTEGER*4

     SUBROUTINE mprecvi( buffer, nitem, src, msg )

     IMPLICIT NONE
     INCLUDE  'mpif.h'
     INTEGER  status(MPI_STATUS_SIZE)
     INTEGER  buffer(*), nitem, src, msg, info

     CALL MPI_RECV( buffer, nitem, MPI_INTEGER, src,
    *               msg, MPI_COMM_WORLD, status, info )

     END

C  MPSENDI: odeslání zprávy, data typu INTEGER*4

     SUBROUTINE mpsendi( buffer, nitem, dest, msg )

     IMPLICIT NONE
     INCLUDE  'mpif.h'
     INTEGER  buffer(*), nitem, dest, msg, info

     CALL MPI_SEND( buffer, nitem, MPI_INTEGER,
    *               dest, msg, MPI_COMM_WORLD, info )

     END

C  MPMCSTI: hromadné odeslání zprávy, data typu INTEGER*4

     SUBROUTINE mpmcsti( buffer, nitem, ids, nids, msg )

     IMPLICIT NONE
     INCLUDE  'mpif.h'
     INTEGER  buffer(*), nids, nitem, ids(*), msg, info, i

     DO 5 i = 1,nids
        CALL MPI_SEND( buffer, nitem, MPI_INTEGER,
    *                  ids(i), msg, MPI_COMM_WORLD, info )
5    CONTINUE

     END

C  MPSTOP: ukončení spolupráce s paralelní knihovnou

     SUBROUTINE mpstop( )

     IMPLICIT NONE
     INCLUDE  'mpif.h'
     INTEGER  info

     CALL MPI_FINALIZE( info )

     END
```

Obrázek 10: Návrh modulů pro posílání zpráv (pro PVM vlevo, pro MPI vpravo).

2.4.2 Sledování běhu programu

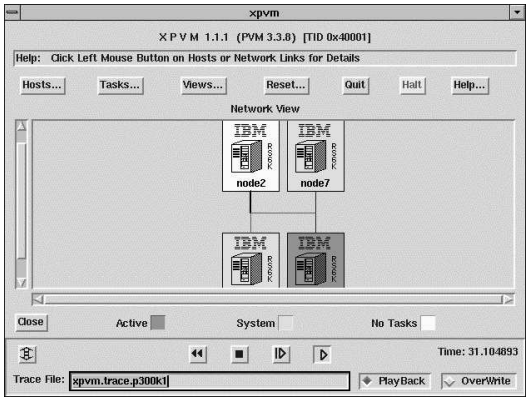
Při vyvíjení paralelního programu je často obtížné určit jeho chování, neboť na něj působí příliš mnoho vlivů současně. Znalost chování je však užitečná pro vyhodnocení běhu programu podle zvolených kritérií a rovněž pro posouzení jeho efektivity. Jedním z možných přístupů k této problematice je použití některého z nástrojů pro sledování běhu programu.

Visualizátor XPVM

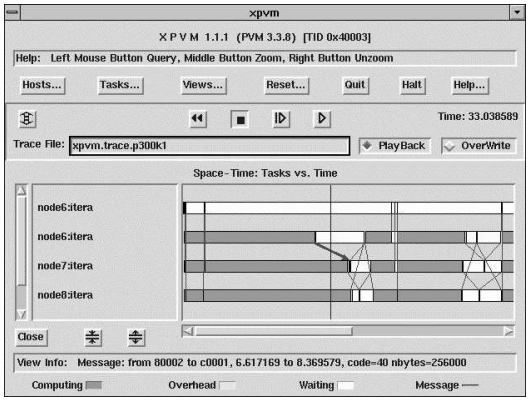
K prakticky běžně užívaným nástrojům patří visualizační program XPVM, což je grafická konzola a monitorovací systém pro PVM. Prostřednictvím systému X-Windows nabízí XPVM grafické rozhraní pro přístup k funkcím uživatelské konzoly PVM, monitorovací systém virtuálního stroje v reálném čase, nástroje pro analýzu zaznamenaných dat a ladící prostředky. Bližší informace o XPVM verze 1.0 můžeme nalézt v [17].

XPVM dovoluje uživateli ovládat virtuální stroj pomocí myši, pouze některé vstupní údaje musí být zadány z klávesnice. Tímto způsobem uživatel může přidávat nebo odebírat hostitelské počítače, spouštět úlohy a řídit jejich běh.

Informace o operacích, které jsou vykonávány virtuálním strojem, na požádání generují[5] systémové knihovny PVM. Tyto informace XPVM dekóduje a ukládá do souboru na pevném disku, který umí znovu načíst a jeho data vyhodnotit. Tato vlastnost umožňuje provozovat XPVM ve dvou režimech: buď slouží ke zobrazení aktuálního stavu programu a jeho změn v reálném čase nebo k analýze programu prostřednictvím přehrávání dat uložených v souboru na disku.



Obrázek 11: Architektura virtuálního stroje znázorněná XPVM.



Obrázek 12: Záznam operací konaných virtuálním strojem v závislosti na čase.

[5] Tuto činnost umožňují systémové knihovny PVM až od verze 3.3.

K nejdůležitějším informacím poskytovaným monitorovacím systémem, jenž je implementován v XPVM, patří znázornění struktury virtuálního stroje včetně stavů uzlů i komunikačních linek (obrázek č. 11) a zobrazení aktuálního stavu všech procesů paralelní aplikace v reálném čase (obrázek č. 12). Visualizace činnosti virtuálního stroje také zahrnuje znázornění všech zpráv předávaných mezi procesy, podrobné i globální prohlížení zaznamenaných dat a dostupnost detailní informace o každé provedené funkci PVM.

XPVM umí zobrazit závislost aktuálního stavu všech procesů paralelní aplikace na čase formou plošného grafu nebo v reálném čase zachycuje aktuální velikost fronty zpráv každého procesu formou sloupcového grafu. Součástí výstupů XPVM jsou také jednoduché (ladící) textové výpisy, které obsahují detailní informace o všech volaných funkcích z knihovny PVM včetně jejich parametrů či návratových kódů. Samozřejmostí je potom shromáždění a zobrazení výstupu paralelního programu.

Monitorování MPI

Jelikož se standard MPI týká pouze rozhraní pro předávání zpráv a nemá jinou vazbu na jeho konkrétní programovou implementaci, je situace ohledně monitorování MPI složitější než v případě PVM. Srovnatelné možnosti jako XPVM nabízí například visualizační program XMPI, který však umí spolupracovat výhradně s LAM *(Local Area Multicomputer)*, což je jedna z rozšířenějších implementací systému pro předávání zpráv založená na MPI. Podrobné informace o XMPI verze 2.2 a LAM verze 6.5.6 můžeme získat například prostřednictvím Internetu z domovské stránky těchto projektů, jejíž adresa je [28].

XMPI je grafické uživatelské prostředí pro spouštění paralelních programů určených k provozu v LAM, pro monitorování stavů jejich procesů i předáváných zpráv. Ke své činnosti využívá ladících schopností prostředí LAM a graficky zpracované informace zobrazuje prostřednictvím systému X-Windows tak, že uživatel má možnost pomocí myši ovládat běhy paralelních programů a zároveň o nich získávat maximum dostupných informací. Výstupy XMPI jsou prakticky stejné jako u XPVM, drobné rozdíly korespondují s odlišnostmi MPI a PVM.

2.4.3 Testování výkonu počítače

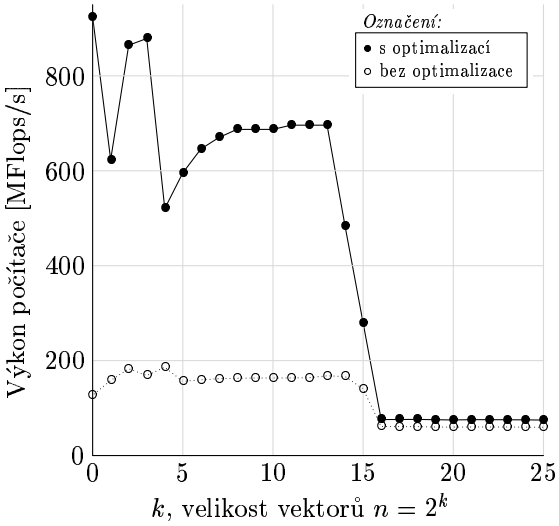
Dosažitelný výkon konkrétního sekvenčního počítače můžeme měřit například pomocí testovacího programu `SCAL`, jehož úplný zdrojový kód je zachycen na obrázku č. 13. Program měří reálnou rychlost procesorů v jednotkách Flops/s (*floating point operations per second*, počet operací v plovoucí řádové čárce za sekundu).

Rychlost procesoru se zjišťuje měřením skutečné doby výpočtu skalárního součinu dvou vektorů reálných čísel, kde je znám počet operací sčítání a násobení. Velikost vektorů lze nastavit parametrem n. Aby nedocházelo k ovlivnění výsledků nepřesným odečítáním příliš krátkých časů, je celý výpočet prováděn

opakovaně. Počet opakování *loops* závisí na n tak, aby se součinem obou čísel zachoval konstantní počet operací v plovoucí řádové čárce, který byl stanoven na hodnotu 2^{29}.

```
      PROGRAM scal

      INTRINSIC  ETime

      INTEGER    n, i, j
      INTEGER    loops, flops

      PARAMETER( n=2**14 )

      REAL       t, t0, tm(2), r
      REAL       a(n), b(n), s, p

      loops = 2**28/n
      flops = 2*n*loops/1.0E6

      DO 10 i = 1,n
         a(i) = 1.0
         b(i) = 2.0
10    CONTINUE

      WRITE(*,*)
      WRITE(*,*) 'SCALAR PRODUCT BENCHMARK'
      WRITE(*,*)
      WRITE(*,*) 'Length of vectors:      ', n
      WRITE(*,*) 'Number of loops:        ', loops
      WRITE(*,*) 'Number of MFlops:       ', flops
      WRITE(*,*)

      CALL ETime( tm, r )
      t0 = tm(1) + tm(2)

      DO 30 j = 1,loops
         s = 0.0
         DO 20 i = 1,n
            s = s + a(i)*b(i)
20       CONTINUE
30    CONTINUE

      CALL ETime( tm, r )
      t = tm(1) + tm(2) - t0

      p = REAL( flops/t )

      WRITE(*,*) 'Control output:        ', s
      WRITE(*,*) 'Computation time [s]:  ', t
      WRITE(*,*) 'Performance [MFlops/s]:', p
      WRITE(*,*)

      END
```

Obrázek 13: Testovací program `SCAL`.



Obrázek 14: Očekávaný výkon každého uzlu klastru LINUX-4.

Poznámky:

- Procesor AMD Athlon 1400 MHz, L1 cache 128 kB, L2 cache 256 kB
- Překlad bez optimalizace:
  `g77-3.0`
- Překlad s optimalizací:
  `g77-3.0 -O3 -march=athlon`

Možnost změny velikosti vektorů je důležitá pro případné odhalení ztráty výkonu počítače v závislosti na rostoucí velikosti zpracovávaných dat, například v důsledku nepříznivých efektů při využívání rychlé vyrovnávací paměti *(cache)* mezi procesorem a operační pamětí.

Na obrázku č. 14 je příklad použití programu `SCAL` na platformě LINUX-4. Uzly klastru byly vybaveny procesory AMD Athlon s taktovací frekvencí 1.4 GHz a dvouúrovňovou rychlou vyrovnávací pamětí o velikostech 128 kB v první úrovni (*L1 cache*) a 256 kB v úrovni druhé (*L2 cache*). Program byl přeložen do spustitelného tvaru bez optimalizace pomocí příkazu `g77-3.0` a s optimalizací příkazem `g77-3.0 -O3 -march=athlon`.

Výsledky testu dokumentují význam použití optimalizačních voleb překladače, které zvyšují efektivitu výsledného kódu až 6×, kdy testovaný počítač dosahoval průměrného výkonu asi 700 MFlops/s a špičkového až 880 MFlops/s. Dále je

vidět několikanásobný pokles výkonu počítače při zpracování dat o velikosti větší než je velikost rychlé vyrovnávací paměti. V takovém případě pracuje optimalizovaný program pouze o 25 % rychleji než program neoptimalizovaný a počítač dosahuje výkonu přibližně 75 MFlops/s.

Ke zjištění dosažitelného výkonu konkrétního paralelního počítače můžeme použít program `SCALP`[6]. Úplný výpis jeho zdrojového kódu pro prostředí MPI zachycuje obrázek č. 15. Program zjišťuje výkon počítače v jednotkách Flops/s měřením skutečné doby paralelního výpočtu skalárního součinu dvou vektorů reálných čísel, kde je znám počet operací sčítání a násobení. Stejně jako v sekvenční verzi se rovněž paralelní výpočet provádí opakovaně tak, aby program celkově vykonal určitý počet operací v plovoucí řádové čárce, stanovený na hodnotu 2^{29}.

V programu lze nastavit velikost vektorů n, které jsou při výpočtu rozděleny na části o stejné velikosti. Počet částí vektorů koresponduje s hodnotou num, což je počet paralelních procesů, jenž výpočet provádějí. Dále je možné v programu zvolit počet interakcí paralelních procesů c, což je výhodné pro sledování změn výkonu paralelního počítače v závislosti na celkovém množství komunikace.

Obrázek č. 16 znázorňuje výkonové charakteristiky klastru LINUX-4 v závislosti na velikosti zpracovávaných dat n a celkovém počtu komunikací c. Program byl přeložen s optimalizací příkazem `g77-3.0 -O3 -march=athlon` a výsledný spustitelný kód běžel na dvou uzlech, $P = 2$. Z výsledků testu je patrný očekávaný pokles výkonu paralelního počítače s rostoucím počtem komunikací paralelních procesů $c = \{8, 64, 512\}$. S nejmenším počtem komunikací počítač dosahoval špičkového výkonu přes 1700 MFlops/s, běžně však kolem 1400 MFlops/s a při zpracování velmi objemných dat pouze 150 MFlops/s.

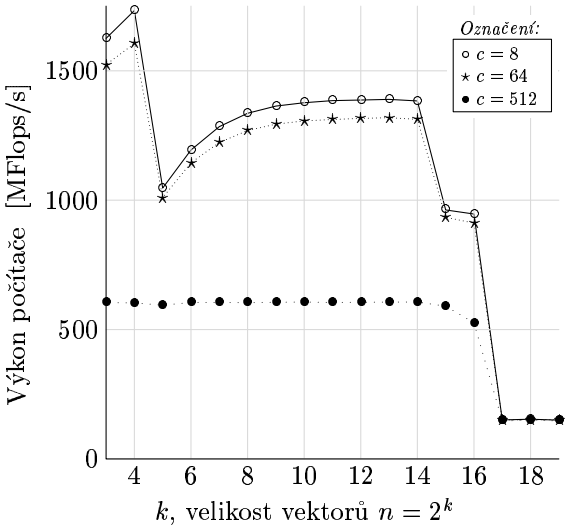
Na obrázku č. 17 jsou zachyceny výkonové charakteristiky klastru LINUX-4 v závislosti na velikosti zpracovávaných dat n a počtu procesorů $P = \{2, 4, 8\}$ využitých při testu. Všechny výpočty byly provedeny s velmi malým počtem komunikací paralelních procesů $c = 8$. Do obrázku byla zaznačena také výkonová charakteristika sekvenčního programu `SCAL`, $P = 1$. Výsledky testu potvrzují očekávaný, přibližně lineární růst výkonu klastru s rostoucím počtem jeho využitých uzlů. Dále je vidět, že mez, kdy výkon počítače prudce poklesne při zpracování dat o velikosti větší než je velikost rychlé vyrovnávací paměti každého uzlu, roste se zvyšujícím se počtem využitých uzlů. Tato příznivá vlastnost přirozeně vyplývá z rozdělení stejného množství dat mezi větší počet uzlů. Pro největší počet uzlů dosahoval počítač špičkového výkonu asi 5000 MFlops/s, běžného výkonu asi 4500 MFlops/s a výkonu 615 MFlops/s při zpracování velmi objemných dat.

Poznamenejme, že v praxi se pro měření výkonu počítače používají testy založené na složitějších kódech, například NAS [32] nebo HPL [31], jenž využívají více různých operací lineární algebry jako jsou součin matice a vektoru, rozklady matic a jiné. Takové obecné testy vykazují reálnější měření výkonu počítače.

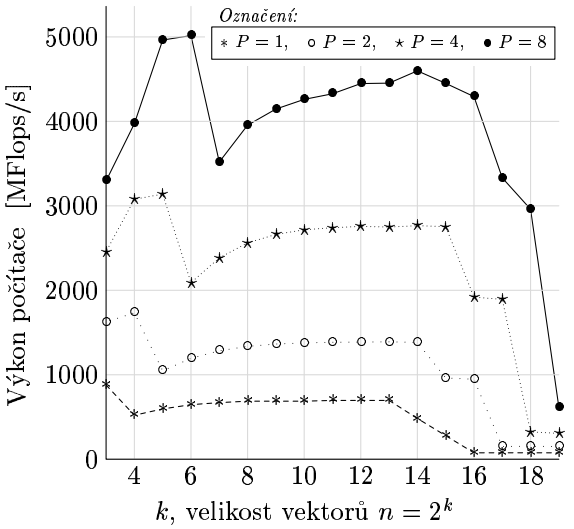
[6] Paralelní obdoba testovacího programu `SCAL`, jehož výpis je zachycen na obrázku č. 13.

```
      PROGRAM scalp

      IMPLICIT           NONE

      INCLUDE            'mpif.h'

      INTEGER            mid, num, info, i, j, k
      INTEGER            n, ns, c, loops, flops

      PARAMETER( n=2**14 )
      PARAMETER( c=2**3 )

      REAL               a(n), b(n), s, sum, p
      DOUBLE PRECISION   t, t0

      CALL MPI_INIT( info )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, mid, info )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, num, info )

      loops = 2**28/(n*c)
      flops = 2**29/1.0E6
      ns = n/num

      sum = 0.0
      DO 10 i = 1,n
         a(i) = 1.0
         b(i) = 2.0
10    CONTINUE

      IF( mid .EQ. 0 ) THEN
        WRITE(*,*)
        WRITE(*,*) 'PARALLEL SCALAR PRODUCT BENCHMARK'
        WRITE(*,*)
        WRITE(*,*) 'Number of processes:    ', num
        WRITE(*,*) 'Number of comm. loops:  ', c
        WRITE(*,*)
        WRITE(*,*) 'Length of vectors:      ', n
        WRITE(*,*) 'Number of loops:        ', loops
        WRITE(*,*) 'Number of MFlops:       ', flops
        WRITE(*,*)
      ENDIF

      CALL MPI_BARRIER( MPI_COMM_WORLD, info )

      t0 = MPI_WTIME()

      DO 50 k = 1,c

         DO 30 j = 1,loops
            s = 0.0
            DO 20 i = 1,ns
               s = s + a(i)*b(i)
20          CONTINUE
30       CONTINUE

         CALL MPI_REDUCE( s, sum, 1, MPI_REAL, MPI_SUM,
     *                    0, MPI_COMM_WORLD, info )

         IF( ( mid .EQ. 0 ).AND.( k .EQ. c ) )
     *     WRITE(*,*) 'Control output:         ', sum

50    CONTINUE

      t = MPI_WTIME() - t0

      p = REAL( flops/t )

      IF( mid .EQ. 0 ) THEN
        WRITE(*,*) 'Computation time [s]:  ', t
        WRITE(*,*) 'Performance [MFlops/s]:', p
        WRITE(*,*)
      ENDIF

      CALL MPI_FINALIZE( info )

      END
```

Obrázek 15: Testovací program SCALP.



Obrázek 16: Test výkonu klastru LINUX-4 programem SCALP pro různé počty komunikací c. Počet využitých uzlů $P = 2$.



Obrázek 17: Test výkonu klastru LINUX-4 programem SCALP pro různé počty využitých uzlů P. Počet komunikací $c = 8$.

Poznámka:

- Ve zdrojovém kódu programu SCALP je pro počet procesorů P, které budou využity při běhu programu, vyhrazena proměnná *num*

3 Řešená problematika

Velkou třídu technických problémů, například posuzování změn podloží zatíženého stavební konstrukcí nebo vývoje napěťových polí při těžbě v dole, matematicky modelujeme pomocí okrajových úloh pružnosti. Řešení úloh tohoto typu založíme na jejich diskretizaci pravidelnou sítí a následné aplikaci metody konečných prvků, která vede obvykle na řešení soustavy lineárních rovnic. Velikost soustavy může být značná, může se pohybovat řádově i v desítkách miliónů rovnic. Taková extrémní velikost potom brání nalezení řešení soustavy použitím některé z metod přímých a proto aplikujeme metodu nepřímou. V praxi se používá metoda sdružených gradientů s předpodmíněním, jejíž různé algoritmické varianty detailně popíšeme v kapitole 4. Na základě těchto algoritmů a s využitím znalostí paralelních výpočtů, s nimiž jsme se seznámili v kapitole 2, programově realizujeme odpovídající řešiče soustav, které důkladně vyzkoušíme na testovacích úlohách.

3.1 Formulace okrajové úlohy pružnosti

Označíme Ω studovanou trojrozměrnou oblast a Γ její hranici, která je rozdělená na vzájemně disjunktní části, $\Gamma = \Gamma_0 \cup \Gamma_1 \cup \mathcal{R}$. Okrajovou úlohu pružnosti formulujeme pomocí soustavy diferenciálních rovnic

$$\sum_j \frac{\partial \tau_{ij}}{\partial x_j} = f_i \qquad \text{v } \Omega \tag{1}$$

a okrajových podmínek

$$u_i = \hat{u}_i \qquad \text{na } \Gamma_0 \ , \tag{2}$$

$$\sum_j \tau_{ij}\nu_j = T_i \qquad \text{na } \Gamma_1 \ . \tag{3}$$

Současně platí

$$\tau_{ij} = \sum_{kl} c_{ijkl} e_{kl} \ , \tag{4}$$

$$e_{kl} = \frac{1}{2}\left(\frac{\partial u_l}{\partial x_k} + \frac{\partial u_k}{\partial x_l}\right) . \tag{5}$$

Uvedené rovnice definují závislosti mezi veličinami po složkách. Uvažujeme trojrozměrný prostor, proto v (1) až (5) budou $i, j, k, l = 1, 2, 3$.

Soustava (1) vyjadřuje rovnováhu sil v oblasti Ω, kde f je působící síla, τ tenzor napětí lineárně svázaný s tenzorem malých deformací e Hookovým zákonem (4) a c konstanty charakterizující materiálové vlastnosti. Rovnice (5) vyjadřuje vztah mezi tenzorem malých deformací e a posunutím u. Řešením okrajové úlohy pružnosti získáme napěťový stav, deformace a posunutí v oblasti Ω. Vztahem

(2) je na části hranice označené Γ_0 předepsáno posunutí $\hat{u}$. Okrajové podmínky tohoto typu nazýváme Dirichletovými. Vztahem (3), kde ν je jednotkový vektor vnější normály k části hranice označené Γ_1, jsou dány povrchové síly T na této části hranice. V tomto případě hovoříme o okrajových podmínkách Neumannova typu.

Diferenciální rovnice (1) platí v každém bodě oblasti Ω, mají lokální charakter. Hlubší fyzikální oprávnění má však jiný způsob vyjádření rovnováhy - integrálními rovnicemi podle tzv. variačních principů pro energii, které se ukazují velmi vhodné jak pro matematickou analýzu, tak pro přibližné řešení fyzikálních úloh. Podle zmíněných principů můžeme okrajovou úlohu pružnosti formulovat variačně: hledáme posunutí $u \in [H^1(\Omega)]^3$ takové, že

$$u - u_0 \in V \ , \qquad V = \{v \in [H^1(\Omega)]^3 : \ v = 0 \text{ na } \Gamma_0\} \ ,$$

$$\int_\Omega \sum_{ijkl} c_{ijkl} \frac{\partial u_i}{\partial x_j} \frac{\partial v_k}{\partial x_l} \, d\Omega = \int_\Omega f v \, d\Omega \ + \int_{\Gamma_1} T v \, dS \qquad \forall v \in V \ . \tag{6}$$

V uvedené formulaci je $u_0 \in [H^1(\Omega)]^3$ posunutí, které nabývá předepsaných hodnot na Γ_0, v je libovolné možné (virtuální) posunutí, které můžeme chápat jako vychýlení z rovnovážné polohy, f je hustota objemových sil a T hustota povrchových sil. Výrazy fv, Tv představují skalární součiny a $H^1(\Omega)$ je Sobolevův prostor. Význam ostatních veličin jsme popsali dříve.

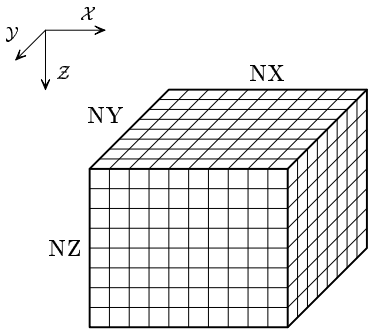
Rovnice rovnováhy (6) bývá často interpretována takto: virtuální práce vnitřních sil je rovna virtuální práci vnějších sil, tedy součtu virtuálních prací sil objemových a povrchových.

Pro podrobnější vysvětlení významu uvedených rovnic a pojmů týkajících se klasické a variační formulace okrajových úloh pružnosti odkazujeme čtenáře především na knihu [6], kde se můžeme seznámit i s dalšími podmínkami a omezeními kladenými na veličiny ve výše uvedených vztazích.

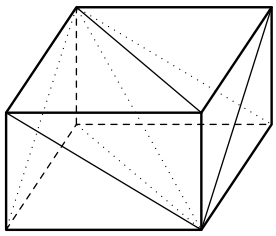
3.2 Diskretizace metodou konečných prvků

Přibližné řešení okrajové úlohy pružnosti můžeme založit na dělení studované oblasti Ω a následné aplikaci metody konečných prvků vycházející z variační formulace úlohy. Tento postup vede na řešení soustavy lineárních rovnic, která je rozsáhlá v případě jemného dělení oblasti Ω. Matice vzniklých lineárních systémů jsou řídké, symetrické, positivně definitní a nazýváme je maticemi tuhosti. Detailně se s metodou konečných prvků a aspekty její aplikace při řešení okrajových úloh můžeme seznámit v knihách [7] a [26].

V dalším textu budeme uvažovat pravidelné dělení oblasti Ω. Každý jednotlivý element vzniklé diskretizační sítě, jenž nemusí být pravoúhlá, bude rozdělený na lineární konečné prvky ve tvaru čtyřstěnu. Příklad dělení oblasti a aplikaci konečných prvků znázorňují obrázky č. 18, 19.



Obrázek 18: Pravidelné dělení pravoúhlé oblasti.



Obrázek 19: Dělení elementu sítě na konečné prvky.

Pravidelné dělení oblasti vede k pravidelnému uspořádání nenulových prvků matice tuhosti. Toto uspořádání odpovídá pravidelné šabloně, kterou lze využít k efektivnímu uložení matice tuhosti a tím ke snížení celkové paměťové náročnosti řešení úlohy. Šablona závisí na parametrech diskretizační sítě, konkrétně na NX, NY, NZ, což jsou počty uzlů sítě po řadě ve směru $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$. Poznamenejme, že NN = NX∗NY∗NZ označíme celkový počet uzlů sítě a ND = 3∗NN celkový počet rovnic výsledné soustavy lineárních rovnic.

3.3 Modulární programový systém GEM

Popsaná diskretizace je základem modulárního programového systému GEM vyvíjeného v Ústavu geoniky AV ČR a určeného pro modelování 3D úloh z geomechaniky. Popis systému GEM lze najít v [30] a literatuře zde uvedené.

Celý výpočetní postup, schématicky zachycený na obrázku č. 20, má čtyři hlavní fáze sestávající z jednotlivých kroků, které jsou reprezentovány jedním nebo několika programy vykonávajícími specifikovanou činnost v závislosti na typu řešené úlohy a variantě metody použité pro její řešení.

Počáteční fázi celého výpočetního postupu nazýváme preprocesor. Součástí preprocesoru jsou programy pro načtení informací o úloze, k nimž patří zejména rozměry a tvar studované oblasti, rozložení materiálů a materiálové vlastnosti, popis zatížení a okrajových podmínek, charakteristika diskretizační sítě, konečných prvků a podobně. Tuto činnost vykonává například program TGRID.

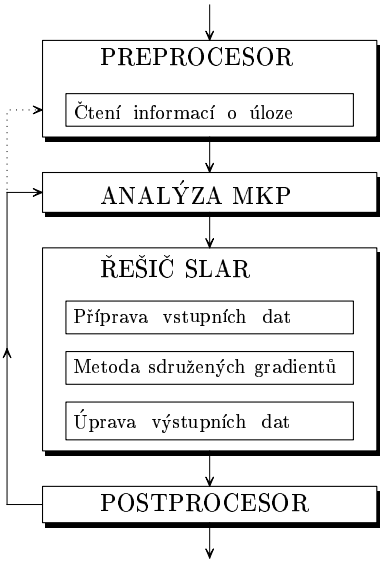
Na preprocesor navazují programy pro analýzu metodou konečných prvků, která vede na sestavení lineárního systému. Tento díl výpočtů realizuje například program SMAT.

Další porci výpočtů obstarají programy určené k řešení soustavy lineárních algebraických rovnic. Jedná se o sekvenční a paralelní řešiče, například SESOL nebo ITERA, které pracují podle některé varianty algoritmu metody sdružených gradientů s předpodmíněním. Do stejné skupiny řadíme i pomocné programy, které slouží k přípravě vstupních dat a úpravě dat výstupních, k distribuci a sběru

dat mezi jednotlivými uzly paralelního počítače a dalším činnostem. Konkrétně k paralelnímu řešiči ITERA patří pomocné programy DSPLIT a SMERGE.

Poslední fázi výpočetního postupu nazýváme postprocesor. Do postprocesoru zahrnujeme programy pro aposteriorní odhady chyb, pro přípravu nové aproximace řešení nebo nových okrajových podmínek pro případný další výpočet, což vykonává například program BCRES, a podobně.

Celý výpočet systémem GEM technicky obvykle realizuje dávka, která volá příslušné programy. Pořadí volání programů přitom odpovídá jednotlivým krokům ze schématu na obrázku č. 20. Ve fázi preprocesoru programy načítají data připravená uživatelem, v dalších fázích potom data vytvořená v předešlých krocích. Běh programů může uživatel řídit prostřednictvím parametrů zapsaných do speciálně pojmenovaných textových souborů, což je výhodné především pro provozování systému GEM na počítačích s dávkovým zpracováním úloh.



Obrázek 20: Schéma výpočtů systémem GEM.

Zdrojové texty všech programů jsou napsány v programovacím jazyku Fortran 77, jehož překladače existují na superpočítačích, pracovních stanicích i osobních počítačích. Původní systém GEM byl přepracován s ohledem na maximální přenositelnost zdrojových kódů programů na různé platformy počítačů a doplněn o systém dávek pro jeho automatizovaný překlad. Tento přirozený vývoj motivoval změnu označení celého systému na PortaGEM (portabilní GEM). Zároveň však zkušenosti získané přenosy systému mezi rozdílnými počítačovými platformami[7] a nové poznatky týkající se aplikovaných numerických metod neustále inspirují autory k dalšímu vylepšování celého systému.

V této práci se budeme zabývat paralelními řešiči systému PortaGEM, které tvoří samostatnou knihovnu pojmenovanou ELPAR. V kapitolách 4 až 6 popíšeme různé varianty algoritmů řešičů i jejich programové realizace. Rovněž ukážeme výsledky testů řešičů provedených na modelových úlohách a stanovíme efektivitu výsledných kódů.

3.4 Sada testovacích úloh z geomechaniky

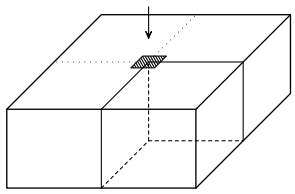
Pro testování řešičů jsme vybrali úlohy z geomechaniky. První dvě z nich jsou akademické modely deformace podloží základu a rovnováhy sil v oblasti, v posledním případě se jedná o matematické modelování reálné situace při dobývání ložiska uranu.

[7]Systém je odzkoušen a může pracovat na platformách IBM SP, SUN a LINUX.

3.4.1 Deformace podloží základu

V úloze, kterou v testech označujeme FOOT, je popsána situace z obrázku č. 21, kdy dochází k deformaci podloží základu, který je modelován silovým zatížením. Oblast je kvádr o rozměrech 100×100×40 metrů, na horní straně uprostřed ve čtvercové oblasti o velikosti 10×10 metrů rovnoměrně zatížený tlakem 2.4 MPa.

Z obrázku je patrná symetrie oblasti i zatížení, proto můžeme uvažovat pouze čtvrtinu původní oblasti o rozměrech 50×50×40 metrů, v ostatních částech bude řešení stejné. Uvažovaná oblast bude zatížená v rohu horní strany na čtvercové ploše o velikosti 5×5 metrů. Vnitřní síly jsou určeny váhou materiálu. Hraniční podmínky představují nulová normálová posunutí a nulové tangenciální tlaky na každé straně kromě horní, kde je vnucený tlak. Materiál je homogenní a izotropní.



Obrázek 21: Deformace podloží základu.

Soustava lineárních rovnic vznikla diskretizací oblasti pravidelnou sítí, která byla jemnější směrem k namáhané oblasti, a aplikací metody konečných prvků. Celkový počet rovnic výsledné soustavy se různil v závislosti na počtu uzlů diskretizační sítě, který byl pro každou variantu úlohy ve všech směrech stejný. Pro jednotlivé varianty úlohy nabýval hodnot 6 až 81 uzlů (s krokem 5). Charakteristiky vybraných variant úlohy jsou shrnuty v tabulce č. 1.

Označení	Počet uzlů sítě	Počet rovnic
FOOT 05 E	06×06×06	648
FOOT 10 E	11×11×11	3 993
FOOT 20 E	21×21×21	27 783
FOOT 40 E	41×41×41	206 763
FOOT 60 E	61×61×61	680 943
FOOT 80 E	81×81×81	1 594 323

Tabulka 1: Vybrané varianty úlohy FOOT-E.

Označení	Počet uzlů sítě	Počet rovnic
FOOT 04	04×04×04	192
FOOT 12	12×12×12	5 184
FOOT 20	20×20×20	24 000
FOOT 40	40×40×40	192 000
FOOT 60	60×60×60	648 000

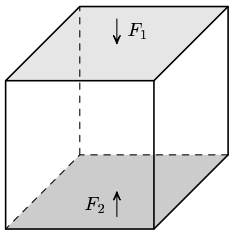
Tabulka 2: Původní varianty úlohy FOOT.

Všechny uvedené varianty úlohy FOOT-E vycházejí z původní úlohy FOOT 40, která označuje řešení stejného problému diskretizovaného jinou sítí. Použitá síť o velikosti 40×40×40 uzlů však až na výjimečné případy neumožňovala odvodit různé varianty úlohy pouze úpravou počtů uzlů sítě v jednotlivých směrech tak, aby se nezměnila celková geometrie úlohy. Tímto způsobem bylo možné vytvořit celkově jen 5 původních variant úlohy, které jsou shrnuty do tabulky č. 2. Potřeba testovat řešiče na úlohách různé velikosti proto zapříčinila nezbytnou úpravu diskretizační sítě originální úlohy FOOT a tím vytvoření nové, škálovatelné testovací úlohy FOOT-E.

3.4.2 Rovnováha sil v oblasti

Pro experimenty s lineárními systémy, jejichž matice tuhosti je singulární, jsme vytvořili modelový příklad rovnováhy sil v oblasti. Tuto úlohu v testech označujeme NEUM.

Studovaná oblast má tvar krychle o rozměrech 100×100×100 metrů a je materiálově homogenní. Celá plocha horní strany je zatížená silou F_1 o velikosti 10.0 MPa. Proti ní působí na celou plochu dolní strany síla F_2, jejíž velikost je dána součtem velikostí působící síly F_1 a tíhy tělesa určené váhou materiálu, $F_2 \sim 19.9$ MPa. Tím je dosaženo rovnováhy sil. Poznamenejme, že na ostatních stranách jsou zadány nulové síly.



Obrázek 22: Úloha pouze se silovými okrajovými podmínkami.

Všechny okrajové podmínky jsou Neumannova typu. Diskretizace úlohy konečnými prvky potom vede na řešení lineárního systému se singulární maticí tuhosti, pro který hledáme zobecněné řešení. K takovému výpočtu je však nutná stabilizace metody sdružených gradientů, kterou popíšeme v dalším textu.

Označení	Počet uzlů sítě	Počet rovnic
NEUM 11	11×11×11	3 993
NEUM 41	41×41×41	206 763

Tabulka 3: Varianty úlohy NEUM.

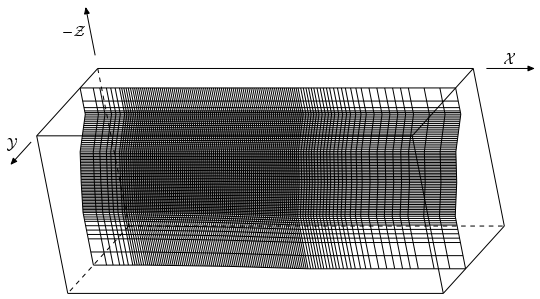
Soustava lineárních rovnic úlohy NEUM vznikla diskretizací oblasti pravidelnou sítí a aplikací metody konečných prvků. Velikost výsledné soustavy závisela na hustotě použité diskretizační sítě. Tabulka č. 3 popisuje dvě varianty úlohy, které jsme pro účely testování řešičů vytvořili.

3.4.3 Těžba v uranové sloji

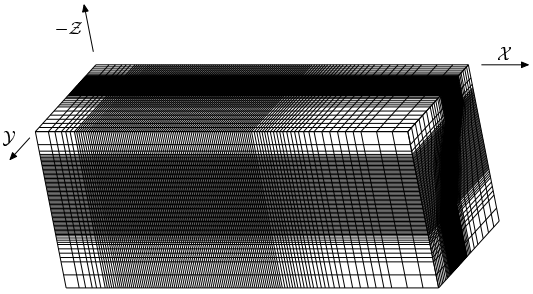
V úloze, kterou v testech označujeme `DORO`, je modelován vývoj napěťových polí při dobývaní ložiska v uranovém dole v lokalitě Dolní Rožínka. Uvažovaná oblast o rozměrech 1430×550×600 metrů je materiálově nehomogenní a je umístěná 700 metrů pod povrchem. Nacházejí se v ní tři vzájemně rovnoběžné uranové sloje. Jejich umístění v oblasti je zachyceno na obrázku č. 23.

Okrajové podmínky jsou zvoleny buď Neumannova typu, které simulují počáteční napěťový stav, nebo Dirichletova typu, které jsou dané posunutími před těžbou a musí být spočítány předem. S detaily konstrukce matematického modelu se můžeme seznámit například prostřednictvím [21].

Postupně se simulují čtyři fáze těžby, které jsou reprezentovány sekvencí výpočtů stejného problému s různými distribucemi materiálů. Sekvence výpočtů `NS` představavuje řešení čtyř problémů s okrajovými podmínkami Neumannova typu, sekvence výpočtů `DS` potom řešení pěti problémů, z nichž první má okrajové podmínky Neumannova typu, ostatní Dirichletova typu.



Obrázek 23: Modelování ložiska uranu ve studované oblasti.



Obrázek 24: Diskretizace oblasti s modelovaným ložiskem uranu.

Diskretizací oblasti pravidelnou sítí, která je zachycena na obrázku č. 24, a metodou konečných prvků vznikly soustavy lineárních rovnic o 3 873 264 neznámých. Poznamenejme, že v případě modelovací sekvence `NS` jsou soustavy singulární, v případě modelovací sekvence `DS` pak regulární s výjimkou prvního kroku.

Singulární soustavy mohou být nekonsistentní vlivem zaokrouhlovacích chyb nebo určitých nevyvážeností, které jsou způsobené různou jednotkovou váhou materiálu ve vytěžených částech. Pro takovou singulární soustavu

$$Au = f \ , \qquad f \notin R(A) \ ,$$

kde $R(A)$ je teoretický obor hodnot matice A, potom hledáme zobecněné řešení $u^\star$ ve smyslu

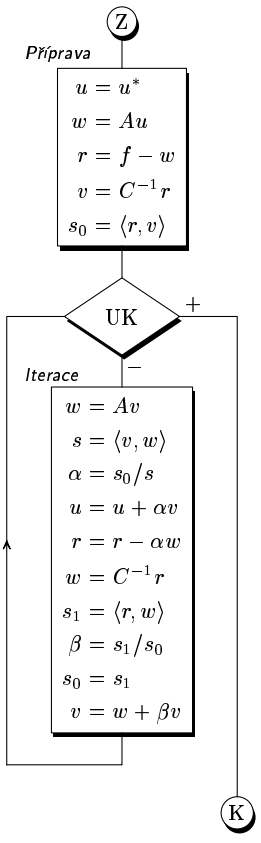
$$\|Au^\star - f\| = \min.$$

4 Iterační řešiče

Výpočetně i časově nejnáročnější částí řešení úlohy pružnosti metodou konečných prvků je řešení výsledné soustavy lineárních algebraických rovnic. V dalším textu proto představíme iterační řešiče lineárních systémů založené na metodě sdružených gradientů s předpodmíněním.

Kromě základního sekvenčního algoritmu uvedeme i jeho zobecněný tvar, různé typy předpodmínění a techniku umožňující řešit singulární úlohy. Rovněž se seznámíme s metodami rozkladu prostoru, které využijeme pro konstrukci účinných předpodmiňovačů a pro návrh algoritmů paralelní metody sdružených gradientů.

4.1 Metoda sdružených gradientů s předpodmíněním



Obrázek 25: Metoda sdružených gradientů s předpodmíněním.

Metoda sdružených gradientů (*conjugate gradient method*, CG) je iterační metoda určená pro řešení (rozsáhlé) soustavy lineárních rovnic

$$Au = f \ , \qquad u, f \in R^{\mathrm{ND}} \tag{7}$$

se symetrickou (ND × ND), positivně definitní maticí soustavy A. Takový lineární systém vzniká například při řešení úlohy pružnosti.

V i-té iteraci metoda sdružených gradientů hledá optimální přibližné řešení u_i na prostoru

$$u_0 + \mathcal{K}_i \ ,$$

kde $\mathcal{K}_i$ je i-tý Krylovův podprostor definovaný

$$\begin{aligned} \mathcal{K}_i &= \mathrm{span}\{r_0, Ar_0, \ldots, A^{i-1}r_0\} \ , \\ r_0 &= f - Au_0 \ . \end{aligned}$$

Metoda nevyžaduje žádné parametry a je paměťově nenáročná.

Rychlost konvergence metody závisí na rozložení vlastních čísel matice soustavy A, přičemž rychlou konvergenci můžeme očekávat v případě čísla podmíněnosti blízkého 1 nebo v případě shluku vlastních čísel. Ke zlepšení konvergence se využívá technika zvaná předpodmínění, kterou si lze představit jako přechod k nové soustavě lineárních rovnic s maticí, která má spektrální vlastnosti lepší než matice původní soustavy A.

Celkové množství práce potřebné pro řešení je u iteračních metod rovno práci pro vykonání jedné iterace vynásobené počtem iterací. U metody sdružených gradientů se v jedné iteraci provádí především součin matice × vektor, dále dvakrát skalární součin vektorů a třikrát násobení vektorů skalárem a sčítání vektorů. Při použití předpodmínění musíme navíc v každé iteraci řešit soustavu s předpodmiňující maticí C.

4.1.1 Základní sekvenční algoritmus

Algoritmus metody sdružených gradientů s předpodmíněním můžeme nalézt například v [10], [11]. V této práci budeme uvažovat jeho upravený tvar znázorněný na obrázku č. 25.

Algoritmus pracuje s pěticí vektorů reálných čísel: hledané řešení u, pravá strana f, residuum r, směr postupu v pro opravu řešení u a pomocný vektor w.

Ukončení iterací

Dvě fáze algoritmu, přípravná a iterační, jsou odděleny podmínkou pro ukončení iterací neboli ukončovacím kritériem UK. Velmi často se používá ukončovací kritérium ve tvaru

$$\frac{\|r\|}{\|f\|} = \sqrt{\frac{\langle r, r\rangle}{\langle f, f\rangle}} \leq \varepsilon \ ,$$

kde požadovanou relativní přesnost ε zadává uživatel. Běžně se setkáme například s hodnotou $\varepsilon = 10^{-4}$.

Jiný přístup k ukončení iterací může být založen na odhadu velikosti relativní chyby řešení, například

$$\frac{\|u - u_k\|}{\|u\|} \leq C(A)\frac{\|f - Au_k\|}{\|f\|} \ ,$$

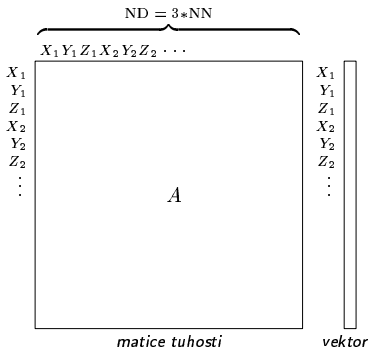
kde $C(A)$ je číslo podmíněnosti matice soustavy A. Hodnota $C(A)$ může být odhadnuta použitím kombinace metody sdružených gradientů a Lanczosovy metody. Uvedený odhad však může být příliš pesimistický, proto je problematika stanovení přesnějších odhadů studována v dalších pracech, například v [24].

Popis datových struktur

S velikostí lineárního systému roste důležitost efektivního uložení jeho datových struktur, zejména matice tuhosti, neboť operace s maticí tuhosti jsou vzhledem k celkovému objemu všech výpočtů během jedné iterace metody sdružených gradientů dominantní a ovlivňují celkovou paměťovou náročnost řešení lineární soustavy.

Pro uložení matice tuhosti se využívají její charakteristické vlastnosti, symetrie a řídkost, a proto stačí ukládat pouze nenulové prvky její horní trojúhelníkové části. Při diskretizaci oblasti pravidelnou sítí má matice tuhosti na každém řádku pravidelně umístěné nenulové prvky, přičemž jejich skutečný počet závisí na typu použitých konečných prvků. Pro případy, kdy je oblast diskretizována způsobem popsaným v oddíle 3.2 (obrázky č. 18, 19), jsou na obrázcích č. 26 a 27 schématicky zachyceny uspořádání prvků datových struktur a detailní struktura matice tuhosti (`X`, `Y`, `Z` označují pozice nenulových prvků odpovídající příslušnému směru posunutí). Poznamenejme, že popis uvedené diskretizace v kontextu původního řešiče `PCG-1` můžeme nalézt v [14].

Matice tuhosti může být uložena do datového souboru například po diagonálách nebo, jako v této práci, po 42 nenulových prvcích každého řádku její horní trojúhelníkové části. Aby se v rutinách, které s maticí pracují, nemuselo ošetřovat zkrácení posledních řádků, jejichž počet je dán poloviční šířkou pásu, ukládají se i tyto řádky v plné délce (doplnění nulami). O poloviční šířku pásu jsou zvětšeny také vektory, které se při výpočtech s maticí používají. Toto zvýšení paměťové náročnosti implementace je poměrně malé, přičemž jsou výrazně zjednodušeny algoritmy, jenž s maticí pracují.



Obrázek 26: Schéma globálního uspořádání prvků datových struktur.

```
          3*NX-2    3*NX*NY-3*NX-2   3*NX*NY-2   3*NX*NY+3*NX-2          ND
      1234567....|.....|.....|.....|.....|.....|.....|.....|.....|.....|

    1: XYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    2:  YZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    3:   ZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    4:    XYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    5:     YZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    6:      ZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0.....0
    7:       XYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0
    8:        YZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0
    9:         ZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0XYZXYZXYZ0...0
    :           :         :           :           :           :           :
ND-5:                                                                  XYZXYZ
ND-4:                                                                   YZXYZ
ND-3:                                                                    ZXYZ
ND-2:                                                                     XYZ
ND-1:                                                                      YZ
  ND:                                                                       Z
```

Obrázek 27: Uspořádání prvků horní trojúhelníkové části matice tuhosti A.

Operace násobení matice × vektor

V rutině realizující operaci násobení matice × vektor, se inicializuje šablona, což je pole indexů patřičné délky, na hodnoty odpovídající skutečným pozicím nenulových prvků v prvním řádku matice. Vždy po zpracování jednoho řádku

matice se šablona aktualizuje na hodnoty odpovídající pozicím nenulových prvků na dalším řádku matice. Stejný způsob práce s maticí tuhosti se používá při předpodmínění.

Samotnou operaci násobení matice × vektor je z hlediska uložení matice během výpočtu možné realizovat dvěma způsoby:

- celá matice se na počátku řešení jednou načte do pole patřičné velikosti v operační paměti a při operacích s maticí se lze odkazovat pouze na prvky tohoto pole, potom odpovídající řešič nazveme paměťově orientovaný,
- matice se vždy čte z datového souboru na disku řádek po řádku, případně po několika řádcích najednou, potom odpovídající řešič charakterizujeme jako diskově orientovaný.

Výhodou prvního přístupu je maximální rychlost provádění operace, jeho nevýhodou je však velká paměťová náročnost uchování matice. Často bývá maximální velikost řešeného problému značně omezena velikostí dostupné operační paměti výpočetního systému. V druhém případě je příznivá paměťová nenáročnost celé operace dosažena pouze za cenu mnohem pomalejšího přístupu k prvkům matice. Tato nevýhodná vlastnost nabyde na významu zejména při práci s velkými datovými soubory, které vznikají při použití jemné diskretizace úlohy pružnosti.

Počáteční aproximace řešení

Výpočet metodou sdružených gradientů je závislý na volbě počáteční aproximace řešení u^*, která ovlivňuje velikost počáteční chyby, ale případně také rychlost konvergence metody. K vhodným a používaným volbám patří nulová počáteční aproximace, $u^* = 0$, nebo výsledek dříve provedeného výpočtu, například produkt jiné numerické metody aplikované na stejnou úlohu.

4.1.2 Pevné a proměnné předpodmínění

Rychlost konvergence metody sdružených gradientů závisí na rozložení vlastních čísel matice soustavy A. Pro zvýšení výkonu iterací a tím i zrychlení konvergence metoda využívá techniku zvanou předpodmínění. Její ideou je vytvoření takové předpodmiňující matice C, aby lineární systém

$$Cu = f$$

byl snadněji řešitelný než původní systém a přitom aby matice C byla blízká matici A. Obecně to znamená levně vytvořit matici C, jejíž inverze by dobře aproximovala inverzi původní matice, $C^{-1} \sim A^{-1}$.

Samotná realizace předpodmínění je možná buď přechodem k transformované soustavě

$$C^{-1}Au = C^{-1}f$$

nebo, jako je tomu v této práci, modifikací residua, kdy je nutné v každé iteraci řešit soustavu s předpodmiňující maticí,

$$Cg = r \ . \tag{8}$$

Budeme uvažovat následující volby matice C:

- $C = I$, kde I je jednotková matice, výpočet probíhá bez předpodmínění,
- $C = D$, kde D je diagonální část matice A, hovoříme o diagonálním předpodmínění,
- $C = (X + L)X^{-1}(X + L)^{\mathrm{T}}$, kde L je dolní trojúhelníková část matice A, X je diagonální matice určená podmínkou $Ce = Ae$, $e = (1, \ldots, 1)^T$, potom hovoříme o předpodmínění typu neúplné faktorizace (*incomplete factorization*, IF). Podrobnější informace o tomto typu předpodmínění můžeme nalézt v [11].

  Poznámka: Aby předpodmiňující matice C výše uvedeného tvaru existovala a byla positivně definitní, musí být $X > 0$. Potom řekneme, že uvedená technika neúplné faktorizace je stabilní. Tato stabilita je zaručena například pro matice $L \geq 0$, $Ae \geq 0$, obecně však zaručena není, což potvrzují zkušenosti při řešení úloh pružnosti a plasticity. Náprava spočívá v nalezení vhodné matice $\tilde{A}$, která aproximuje matici A a jejíž neúplná faktorizace je stabilní. Pro úlohy pružnosti může být vhodným kandidátem $\tilde{A}$ matice, kterou získáme z matice A odstraněním spojů, jenž odpovídají uzlovým posunutím v různých směrech. Více informací lze nalézt v [12] a literatuře zde uvedené.

S dalšími variantami předpodmínění se seznámíme v oddíle 4.2, kde budou popsány předpodmiňovače založené na metodách rozložení prostoru a které lze zároveň využít při paralelní implementaci metody sdružených gradientů.

Předpodmiňovač v metodě sdružených gradientů by měl být reprezentován lineárním, symetrickým, positivně definitním zobrazením. Pokud zobrazení představující předpodmínění tyto vlastnosti splňuje, například diagonální předpodmínění nebo neúplná faktorizace, hovoříme o pevném předpodmínění. V opačném případě hovoříme o proměnném předpodmínění, například pokud soustavu s předpodmiňující maticí řešíme nepřesně vnitřními iteracemi do určité přesnosti ε^*, obvykle $\varepsilon^* = 10^{-1}$.

4.1.3 Algoritmus zobecněné metody

Metoda sdružených gradientů s proměnným předpodmíněním může pracovat, i když předpodmínění není reprezentováno lineárním, symetrickým, positivně definitním zobrazením. Dojde však k porušení A-ortogonality směrů postupu při

hledání řešení, což může vést k destabilizaci celé metody. Náprava spočívá v dodatečné A-ortogonalizaci směrů postupu a takovou metodu sdružených gradientů nazýváme zobecněnou. Blíže se s touto metodou můžeme seznámit prostřednictvím [29].

Zápis upraveného algoritmu ve tvaru vhodném k implementaci na počítači je na obrázku č. 28. Dodatečná A-ortogonalizace může být úplná nebo částečná podle toho, zda se korekce nového směru postupu počítá vůči všem předešlým směrům postupu nebo pouze k několika posledním z nich. Celkový počet směrů postupu pro dodatečnou A-ortogonalizaci je v algoritmu označen m.

$$
\begin{aligned}
u_0 &= \hat{u} \\
r_0 &= f - Au_0 \\
v_0 &= g_0 = G(r_0) \\
\sigma_0 &= \langle g_0, r_0 \rangle \\
&\textbf{for } i = 0, 1, \ldots \textbf{ until } \|r_i\| \leq \varepsilon \, \|f\| \textbf{ do} \\
&\qquad w_i = Av_i \\
&\qquad \gamma_i = \langle v_i, w_i \rangle \\
&\qquad \alpha_i = \sigma_i / \gamma_i \\
&\qquad u_{i+1} = u_i + \alpha_i v_i \\
&\qquad r_{i+1} = r_i - \alpha_i w_i \\
&\qquad g_{i+1} = G(r_{i+1}) \\
&\qquad \sigma_{i+1} = \langle g_{i+1}, r_{i+1} \rangle \\
&\qquad v_{i+1} = g_{i+1} \\
&\quad \textbf{for } k = min\{i+1, m\}, \ldots, 1 \textbf{ do} \\
&\qquad\quad \beta_{i+1}^k = \langle g_{i+1}, (r_{i+2-k} - r_{i+1-k}) \rangle / \sigma_{i+1-k} \\
&\qquad\quad v_{i+1} = v_{i+1} + \beta_{i+1}^k v_{i+1-k} \\
&\quad \textbf{end} \\
&\textbf{end}
\end{aligned}
$$

Obrázek 28: Algoritmus zobecněné metody sdružených gradientů s předpodmíněním.

Ke své činnosti vyžaduje algoritmus uložené $2m+3$ vektory: u pro u_i, g pro g_i, w pro w_i, $v[1:m]$ a $r[1:m]$ pro v_j a r_j, $j = i+1-m, \ldots, i$, kde i označuje aktuální iteraci. Modifikací vztahu pro výpočet β_{i+1}^k, tj.

$$\beta_{i+1}^k = -\langle Ag_{i+1}, v_{i+1-k} \rangle / \gamma_{i+1-k} \ ,$$

mohou být paměťové nároky algoritmu sníženy na celkově $m+3$ vektory, ale v každé iteraci by se musela navíc vykonávat další drahá operace matice $\times$ vektor.

Při řešení nepříliš špatně podmíněné úlohy zobecněnou metodou zpravidla postačuje k její stabilizaci částečná dodatečná A-ortogonalizace směrů postupu, obvykle $m=k=1$. Taková varianta metody při srovnání s algoritmem na obrázku č. 25 vyžaduje navíc uložení jednoho vektoru a výpočet jednoho dodatečného skalárního součinu vektorů v každé iteraci.

4.1.4 Projekce pro řešení singulárních úloh

Diskretizací 3D úloh pružnosti s výhradně Neumannovými okrajovými podmínkami metodou konečných prvků vznikají soustavy lineárních rovnic se singulární maticí tuhosti. Příkladem mohou být úlohy `NEUM` a `DORO`, jejichž charakteristiky jsme uvedli v oddílech 3.4.2, 3.4.3. Nyní popíšeme úpravu metody sdružených gradientů, která ji umožní nalézt zobecněné řešení singulárního lineárního systému, pokud takové řešení existuje.

Označme $N(A)$ nulový prostor matice A. Nechť $R(A)$ je teoretický obor hodnot matice A, který je vzhledem k její symetrii komplementární k nulovému prostoru,

$$R^{\mathrm{ND}} = N(A) \oplus R(A) \ .$$

Připomeneme, že ND je celkový počet rovnic lineárního systému. K zachování stability metody sdružených gradientů při výpočtu dopomůže projekce

$$P: \ \bar{r} \mapsto r \ , \qquad \bar{r} \in R^{\mathrm{ND}} \ , \qquad r \in R(A) \ ,$$

kterou aplikujeme v každé iteraci po výpočtu residua

$$\left.\begin{array}{rcl} \bar{r} & = & r - \alpha w \\ r & = & P(\bar{r}) \end{array}\right\} \quad r \perp N(A) \ .$$

U úloh pružnosti tvoří jádro nulového prostoru matice tuhosti šestice vektorů $p_1, \ldots, p_6$, které odpovídají tuhým pohybům,

$$N(A) = \mathrm{span}\{\, p_1, \ \ldots, \ p_6 \,\} \ .$$

Při uspořádání stupňů volnosti odpovídající separovaným složkám posunutí, jenž bude popsáno v dalším textu, mají vektory jádra následující tvar,

$$p_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} , \qquad p_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} , \qquad p_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} ,$$

$$p_4 = \begin{pmatrix} 0 \\ -x_3 \\ x_2 \end{pmatrix} , \qquad p_5 = \begin{pmatrix} -x_3 \\ 0 \\ x_1 \end{pmatrix} , \qquad p_6 = \begin{pmatrix} -x_2 \\ x_1 \\ 0 \end{pmatrix} ,$$

kde 1 a 0 jsou jednotkový a nulový vektor délky ND/3, dále potom x_1, x_2, x_3 jsou vektory po řadě $\mathcal{X}$-ových, $\mathcal{Y}$-ových a $\mathcal{Z}$-ových souřadnic uzlů použité diskretizační sítě. To znamená, že

$$r = \bar{r} - \sum_{j=1}^{6} \alpha_j p_j \ , \qquad \alpha_j \in R \ ,$$

kde koeficienty α_j určíme z podmínek

$$\langle r, p_j \rangle \;=\; 0 \;, \qquad j = 1, \ldots, 6 \;.$$

což vede na řešení soustavy šesti lineárních rovnic

$$G\alpha = \bar{r}^* \;,$$

Matice G je symetrická a jejími prvky jsou součiny $p_i\, p_j$, $i, j = 1, \ldots, 6$. Prvky pravé strany $\bar{r}^*$ jsou součiny $r\, p_j$, $j = 1, \ldots, 6$.

Je zřejmé, že matici G můžeme vyrobit hned na začátku algoritmu, neboť ta se v průběhu výpočtu nemění. Naopak prvky vektoru $\bar{r}^*$ musí být aktualizovány vždy po výpočtu vektoru $\bar{r}$. Po získání vektoru koeficientů α můžeme aplikovat projekci P.

4.2 Metody rozložení prostoru `SD`

Obecnou třídu numerických metod konstruovaných podle společného schématu, kterým je rozklad prostoru, v němž hledáme řešení, nazýváme metody rozložení prostoru (*space decomposition methods*). Do této třídy patří například metody využívající separaci složek posunutí, Schwarzovy metody rozkladu oblasti s překrytím, metody kompozitních sítí a podobně. V poslední době se metody rozložení prostoru dostávají do popředí zájmů, neboť dovolují paralelizaci iteračních metod a zároveň také konstrukci účinných předpodmiňovačů.

Uvažujme řešení okrajové úlohy pružnosti, která je variačně formulovaná na prostoru $\mathcal{V}$ metodou konečných prvků. Nechť V je podprostorem $\mathcal{V}$ s konečnou dimenzí. Báze $\{\phi_i\}_{i=1}^{ND}$ prostoru V definuje izomorfismus prostorů V a prostoru reálných algebraických vektorů R^{ND}. Aproximace řešení úlohy vycházející z metody konečných prvků potom bude mít tvar

$$u_V = \sum_{i=1}^{ND} u_i \phi_i \;, \tag{9}$$

kde u_i jsou složky vektoru $u \in R^{\mathrm{ND}}$. Tento vektor může být určen z řešení lineárního systému (7), jenž splňuje vlastnosti uvedené v oddíle 4.1, metodou sdružených gradientů s předpodmíněním. Její algoritmus vykonává v i-té iteraci následující operace:

$$\begin{aligned}
&\vdots \\
u_{i+1} &= u_i + \alpha_i v_i \\
r_{i+1} &= r_i - \alpha_i w_i \\
g_{i+1} &= G(r_{i+1}) \\
&\vdots
\end{aligned}$$

Zde je předpodmínění zapsáno v obecném tvaru jako operace G, která počítá pseudoresium g k získání vektoru, který by se nacházel ve směru blíže chybě $A^{-1}r$ než residum r.

Předpodmiňovač G může být založen na rozkladu prostoru V ve tvaru

$$V = \sum_{k=1}^{m} V_k \ ,$$

přičemž podprostory $V_k \subset V$, nemusejí nutně být lineárně nezávislé. Nechť prostor V_k je izomorfní s prostorem R^{ND_k}. Potom lze konstruovat matice přechodu I_k, R_k, jenž reprezentují operace:

- $I_k : R^{\mathrm{ND}_k} \to R^{\mathrm{ND}}$ je prodloužení z k-tého podprostoru dané inkluzí $V_k \subset V$,
- $R_k : R^{\mathrm{ND}} \to R^{\mathrm{ND}_k}$ je restrikce neboli omezení na k-tý podprostor, $R_k = I_k^T$.

Podproblémům na podprostorech potom odpovídají matice $A_k = R_k A I_k^T$, které jsou opět symetrické a positivně definitní.

Celá třída předpodmiňovačů založených na metodách rozkladu prostoru může být představena pomocí obrázku č. 29, který zachycuje algoritmus pro výpočet pseudoresidua g.

$g^0 = 0$
for $k = 1, \ldots, m$ **do**
  $g^k = g^{k-1} + I_k A_k^{-1} R_k z^k$
end
$g = g^m$

Obrázek 29: Algoritmus předpodmiňovače založeného na metodách rozkladu prostoru.

Výběr vektoru z^k ovlivňuje typ předpodmiňovače:

- Volba $z^k = r$ dává aditivní předpodmiňovač, který je snadno paralelizovatelný a lze jej v metodě sdružených gradientů použít přímo, neboť je symetrický a positivně definitní.
- Volba $z^k = r - Ag^{k-1}$ dává multiplikativní předpodmiňovač, který je reprezentován nesymetrickým lineárním zobrazením. K získání symetrického, positivně definitního předpodmiňovače je nutné v algoritmu provést korekce z podprostorů také v obráceném pořadí, což lze učinit změnou mezí cyklu na $k = 1, \ldots, m-1, m, m-1, \ldots, 1$.

  Poznamenejme, že tento předpodmiňovač nelze přímo paralelizovat.

Operaci $w_k = A_k^{-1} v_k$, která se objevuje ve všech předpodmiňovačích založených na metodách rozkladu prostoru, můžeme řešit například neúplnou faktorizací, což dává pevné předpodmínění, nebo vnitřními iteracemi do požadované přesnosti, což charakterizuje proměnné předpodmínění.

V případě multiplikativního nebo proměnného předpodmiňovače není předpodmínění reprezentováno lineárním, symetrickým, positivně definitním zobrazením a metoda sdružených gradientů nemusí vůbec konvergovat. Potom je nutné při výpočtu použít algoritmus zobecněné metody s dodatečnou A-ortogonalizací směrů postupu. Takový algoritmus jsme popsali v oddíle 4.1.3.

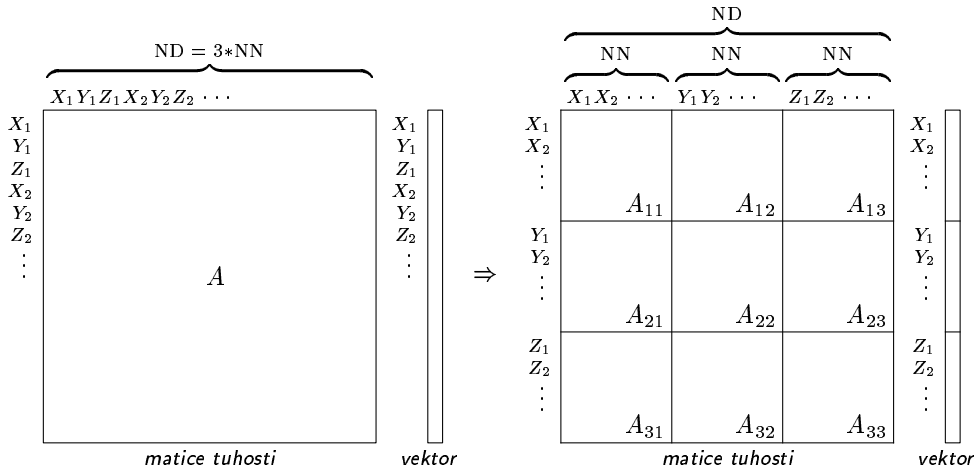
4.2.1 Předpodmínění separací složek posunutí `DiD`

První metodou rozkladu prostoru, kterou se budeme zabývat v této práci, je separace složek posunutí. Poprvé byla tato metoda představena v [5], [12] a její aplikace při paralelních výpočtech byla popsána v [19], [20].

V řešení okrajové úlohy pružnosti (9) reprezentují stupně volnosti u_i uzlová posunutí ve směrech, jenž odpovídají souřadnicovým osám. Odtud lze definovat separaci složek posunutí

$$V = V_{\mathcal{X}} + V_{\mathcal{Y}} + V_{\mathcal{Z}} \ ,$$

kde V_k, $k = \mathcal{X}, \mathcal{Y}, \mathcal{Z}$ je podprostor vektorů, které mají nenulové stupně volnosti ve směru posunutí k. Při uvažování datových struktur v kompaktním tvaru, představeném v oddíle 4.1, je separace složek posunutí docílena přeuspořádáním prvků matice tuhosti a vektorů do ucelených bloků podle směru posunutí. Tuto situaci schématicky zachycuje obrázek č. 30.



Obrázek 30: Schéma separace složek posunutí.

Separace složek posunutí nyní umožňuje představit některé aproximace matice A, jenž dovolují konstrukci účinného předpodmiňovače pro metodu sdružených gradientů. První z nich, matice B,

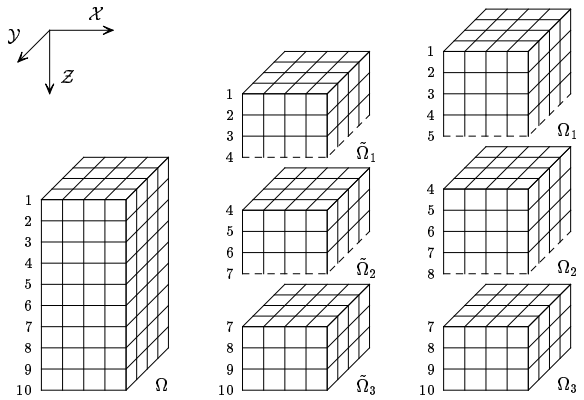
$$B = \begin{bmatrix} A_{11} & & \\ & A_{22} & \\ & & A_{33} \end{bmatrix},$$

je tvořena diagonálními bloky matice A. V této práci je matice B využita při realizaci proměnného předpodmínění, označeného DiD-IE, kdy se operace s inverzní maticí B^{-1} řeší nepřesně iteracemi nové, vnitřní metody sdružených gradientů, které pracují se submaticemi A_{ii}^{-1}, $i = 1, 2, 3$.

Další aproximací matice A je matice C, jenž vychází z matice B, každý blok A_{ii} je však nahrazen neúplnou faktorizací. Připomeneme, že její popis byl v oddíle 4.1.2. V této práci je matice C využita při realizaci pevného předpodmínění, které bude označováno DiD-IF.

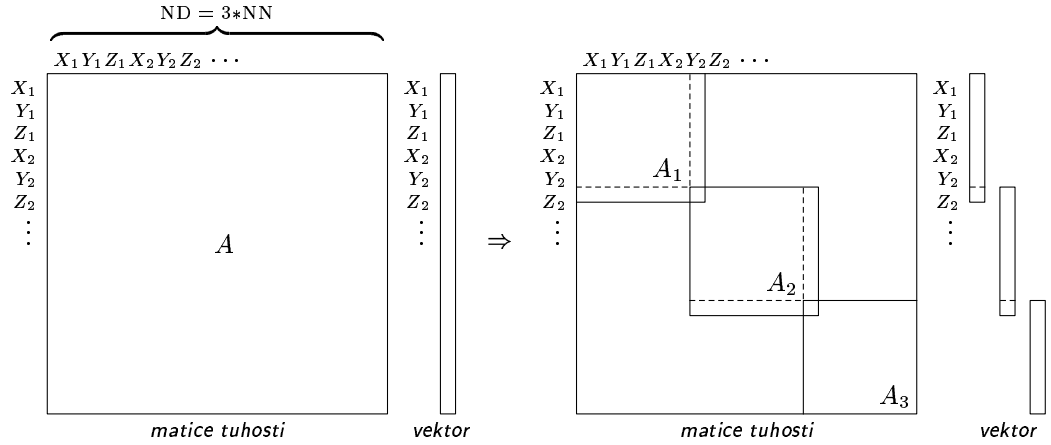
4.2.2 Předpodmínění rozkladem oblasti s překrytím DD

Uvažujme řešení okrajové úlohy pružnosti na 3D oblasti Ω, které jsme popsali v oddíle 3.2, a dále speciální rozdělení oblasti Ω na nepřekrývající se podoblasti $\tilde{\Omega}_i$, $i = 1, \ldots, m$ (m je celkový počet podoblastí), ve směru odpovídajícímu souřadnicové ose $\mathcal{Z}$ tak, že každý uzel sítě bude náležet právě do jedné z podoblastí.



Obrázek 31: Rozložení oblasti na překrývající se podoblasti.

Rozšířením podoblasti $\tilde{\Omega}_i$ o navazující části podoblastí $\tilde{\Omega}_{i-1}$ a $\tilde{\Omega}_{i+1}$ získáme překrývající se podoblasti Ω_i, $i = 1, \ldots, m$. Rozsah překrytí podoblastí OF odpovídá jedné nebo několika vrstvám uzlů diskretizační sítě a má vliv na rychlost řešení metody. V případě rozsahu překrytí podoblastí většího než 1 je vhodné samotný proces překrývání podoblastí provádět tak, že se každá podoblast nejprve rozšíří na konci o jednu vrstvu uzlů z podoblasti následující, potom se rozšíří na začátku o jednu vrstvu uzlů z podoblasti předcházející a celý postup se opakuje.



Obrázek 32: Schéma rozkladu oblasti.

Výjimkami jsou podoblasti Ω_1 a Ω_m, které překrýváme pouze na konci, resp. na začátku podoblasti.

Pokud je tato metoda rozkladu prostoru využita pro paralelizaci metody sdružených gradientů s předpodmíněním, například podle popisu v oddíle 4.3.2, je rovnoměrné rozdělení oblasti Ω na překrývající se podoblasti důležité pro vyvážení zátěže jednotlivých procesů paralelního programu.

Obrázek č. 31 prakticky ukazuje případ, kdy oblast Ω rozdělíme nejprve na tři nepřekrývající se podoblasti $\tilde{\Omega}_1$, $\tilde{\Omega}_2$, $\tilde{\Omega}_3$, které následně rozšíříme o jednu vrstvu uzlů na překrývající se podoblasti Ω_1, Ω_2, Ω_3. Schéma rozkladu matice tuhosti a všech vektorů je zachyceno na obrázku č. 32. Jednotlivým nepřekrývajícím se podoblastem $\tilde{\Omega}_i$ odpovídají submatice $\tilde{A}_i$, jejichž ohraničení je v obrázku pouze naznačeno čárkovanou čárou, a s překrývajícími se podoblastmi Ω_i korespondují submatice A_i.

Rozklad oblasti Ω,

$$\Omega = \bigcup_{i=1}^{m} \Omega_i \ ,$$

zároveň indukuje odpovídající rozklad prostoru V na podprostory V_i,

$$V = \sum_{i=1}^{m} V_i \ , \qquad V_i = \{v \in V : \ v = 0 \text{ in } \Omega \setminus \Omega_i\} \ .$$

Popsaný rozklad oblasti umožňuje vytvořit podobné aproximace matice A, jako tomu bylo v případě separace složek posunutí, a využít je ke konstrukci pevného nebo proměnného předpodmínění pro metodu sdružených gradientů. Matice B ve tvaru

$$B = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_m \end{bmatrix} \ ,$$

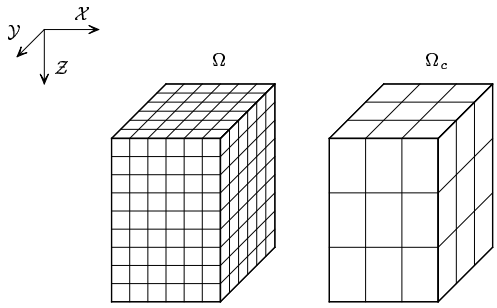
tvořená diagonálními bloky může být základem další aproximace matice A tak, že bloky A_i, $i = 1, \ldots, m$ jsou nahrazeny odpovídající neúplnou faktorizací. Výsledek označíme jako matici C.

Matici B lze přímo využít při realizaci proměnného předpodmínění a matici C k realizaci předpodmínění pevného. Z praktických testů se však ukázalo, že takto konstruovaný proměnný předpodmiňovač není účinný tolik jako předpodmiňovač pevný a proto v této práci bude zahrnut pouze předpodmiňovač pevný, v testech označený jako DD.

Poznamenejme, že metody rozkladu oblasti byly poprvé představeny v roce 1870 německým matematikem Schwarzem[8], po kterém byla tato třída numerických metod posléze pojmenována. V současné době lze podrobné informace o metodách rozkladu oblasti získat v mnoha publikacích, například v [18].

4.2.3 Dvouúrovňové předpodmínění rozkladem oblasti

Efektivita předpodmiňovače založeného na metodě rozkladu oblasti se zvyšuje se rostoucím překrytím podoblastí OF a naopak klesá s rostoucím počtem podoblastí. Tato nevýhoda může být odstraněna a efektivita celého předpodmiňovače zvýšena zahrnutím globální informace reprezentované řešením stejného problému diskretizovaného hrubší sítí. Příklad hrubé sítě je zachycen na obrázku č. 33.



Obrázek 33: Hrubá síť.

Explicitní hrubá síť

Rozklad oblasti Ω, popisovaný v oddíle 4.2.2, je rozšířen o podprostor V_c, $V_c \subset V$, který koresponduje s diskretizací původního problému hrubou sítí reprezentovanou podoblastí Ω_c. Matici tuhosti, která odpovídá podoblasti Ω_c, označíme A_c a budeme předpokládat, že v ní jsou zohledněny Dirichletovy okrajové podmínky. Matice A_c má stejnou strukturu i vlastnosti jako původní matice A, proto pro práci s prvky matice a jejich uložení do datového souboru budou platit stejná pravidla.

Pokud je původní síť zjemněním hrubé sítě (vnořená nebo též vložená hrubá síť, jejíž uzly odpovídají uzlům sítě jemné), budou restrikce R_c a prodloužení I_c, jenž realizují přechod mezi jemnou a hrubou sítí, jednoduše definovány inkluzí $V_c \subset V$, v opačném případě (nevnořená hrubá síť) potom budou dány interpolací hodnot. V této práci budeme uvažovat druhý případ, kdy je přechod mezi jemnou a hrubou sítí dán lineární interpolací hodnot.

[8]Pro řešení eliptické okrajové úlohy na oblasti, která vznikla spojením dvou podoblastí s jednoduchou geometrií, využil Schwarz střídavě řešení původního problému omezeného vždy pouze na jednu z podoblastí.

Nyní můžeme definovat dvouúrovňový aditivní předpodmiňovač vztahem

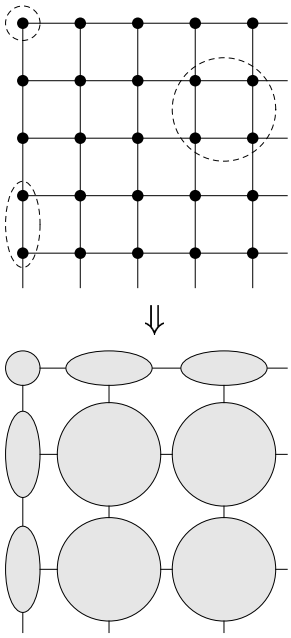
$$w = \left(I_c A_c^{-1} R_c + \sum_{i=1}^{m} I_i A_i^{-1} R_i \right) r \ .$$

Člen, který vstupuje do předpodmiňovače z hrubé sítě, můžeme zařadit nejen aditivně, ale také multiplikativně. V takovém případě vztahy

$$\begin{aligned} w &= I_c A_c^{-1} R_c r \\ w &= w + \sum_{i=1}^{m} I_i A_i^{-1} R_i (r - Aw) \end{aligned}$$

popisují dvouúrovňový hybridní nesymetrický předpodmiňovač. Poznamenejme, že předpodmiňovač můžeme také zkonstruovat jako symetrický, avšak chování metody sdružených gradientů se prakticky nezmění. Z hlediska výpočetní náročnosti jedné iterace metody sdružených gradientů je proto výhodnější použít nesymetrický předpodmiňovač.

Výhodou metody sdružených gradientů s hybridním předpodmiňovačem je až o polovinu nižší počet iterací potřebných k dosažení požadované přesnosti nalezeného řešení než s aditivním předpodmiňovačem. Praxe nám potvrzuje, že aplikace hybridního předpodmiňovače je výhodnější i přes skutečnost, že každá iterace s aditivním předpodmiňovačem má nižší výpočetní náročnost, neboť řešení na hrubé síti lze vykonávat paralelně s výpočty na podoblastech a odpadá nutnost vykonávání drahé operace násobení matice tuhosti × vektor navíc. Při testování řešičů budeme paralelní algoritmus `DD` pracující s hrubou sítí označovat CG.



Obrázek 34: Agregace uzlů na fragmentu horní strany diskretizační sítě.

Agregovaná hrubá síť

V mnoha případech modelování reálných problémů z oblasti geomechaniky nelze explicitní hrubou síť vytvořit, případně je výpočetně drahé vytvoření nebo vykonávání přechodu mezi jemnou a hrubou sítí, realizovaného maticemi R_c, I_c. Proto budeme uvažovat jiný podprostor V_c, konstruovaný z prostoru V agregací.

Hrubá síť, korespondující s podprostorem V_c, vznikne shlukováním neboli agregací uzlů původní diskretizační sítě do uzlů větších. Proces agregace probíhá odděleně pro uzly ležící na hranici oblasti Ω a pro uzly uvnitř této oblasti. Počty uzlů jemné sítě, agregované do jednoho uzlu sítě hrubé, mohou být odlišné

pro každý ze směrů souřadnicových os. Tyto počty budeme v dalším textu nazývat agregační faktory a označovat je AF. Obrázek č. 34 zachycuje na fragmentu horní strany diskretizované oblasti proces shlukování s AF = 2 v každém směru.

Z hlediska praxe proces agregace znamená, že sčítáním příslušných řádků a sloupců v původní matici tuhosti A vytváříme agregovanou matici A_c, která má stejnou strukturu i vlastnosti jako původní matice. Poznamenejme, že stejnou agregací je dán také přechod R_c mezi hodnotami z jemné na hrubou síť. Opačný přechod, matice I_c, je dána prostou distribucí hodnot z uzlů hrubé sítě do odpovídajících uzlů sítě jemné.

Samotná implementace předpodmiňovače s agregovanou hrubou sítí, dále v testech označovaného ACG, se neliší od implementace předpodmiňovače s explicitní hrubou sítí, můžeme realizovat jeho aditivní i hybridní verzi. Ve druhém případě však obvykle při jeho praktickém užití v metodě sdružených gradientů dochází k porušení A-ortogonality směrů postupu při hledání řešení, jehož důsledkem je destabilizace celé metody. Náprava spočívá v dodatečné A-ortogonalizaci směrů postupu, kterou jsme popsali v oddíle 4.1.3 o zobecněné metodě sdružených gradientů.

4.3 Paralelní metoda sdružených gradientů

V dalším textu popíšeme dva algoritmy paralelní metody sdružených gradientů s předpodmíněním. Algoritmy jsou založeny na separaci složek posunutí a na rozkladu oblasti s překrytím. Tyto metody rozložení prostoru, popsané v oddíle 4.2, dovolily přirozenou datovou dekompozici řešeného problému a také konstrukci účinných paralelních předpodmiňovačů.

4.3.1 Algoritmus pro separaci složek posunutí `DiD`

Algoritmus paralelní metody sdružených gradientů s předpodmíněním, založený na separaci složek posunutí, budeme stejně jako v předchozím textu označovat `DiD` podle využité metody rozložení prostoru.

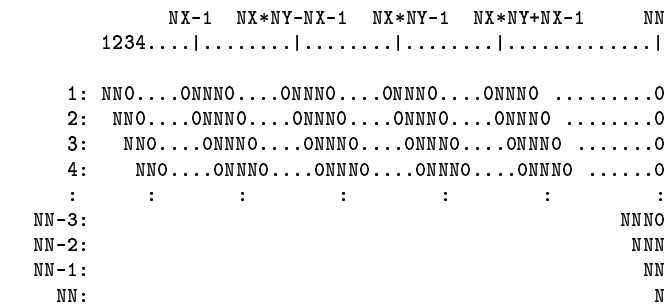
Popis datových struktur

Přeuspořádáním prvků matice tuhosti a vektorů docílíme separaci složek posunutí uzlů sítě do ucelených bloků podle směru posunutí. Přeuspořádání bylo schématicky zachyceno na obrázku č. 30 v oddíle 4.2.1.

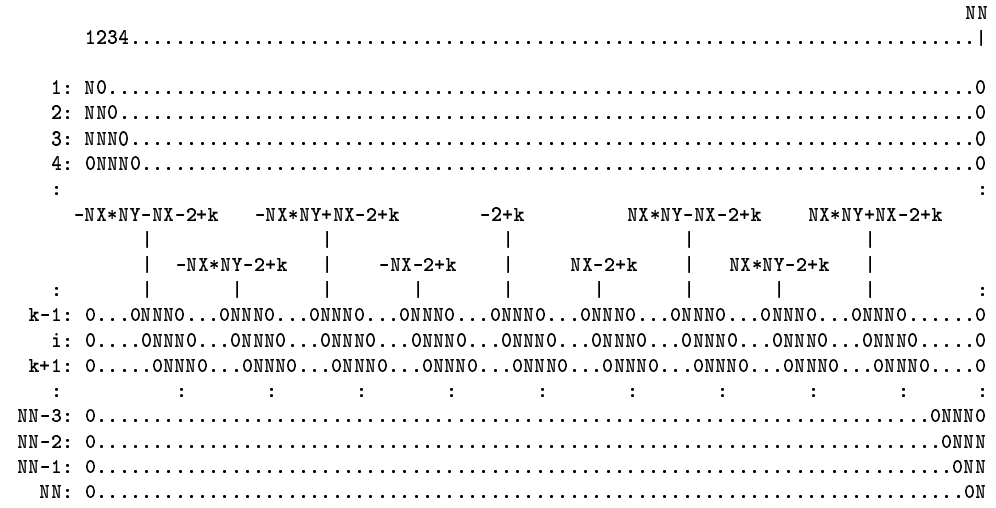
Symetrie matice tuhosti $A = A^T$ se přeuspořádáním prvků zachová, symetrické budou také matice A_{ii}, $i = 1, 2, 3$. Pro ostatní matice A_{ij}, $i, j = 1, 2, 3$, $i \neq j$ bude platit $A_{ij} = A_{ji}^T$. Detailní uspořádání prvků obou typů submatic zachycují obrázky č. 35 a 36 (`N` označuje pozici nenulového prvku).

Submatice A_{ij}, $i, j = 1, 2, 3$, $i \leq j$ je vhodné ukládat (na pevný disk) do samostatných souborů, neboť pokud více procesů paralelní aplikace pracuje s maticí tuhosti, mohou být data na paralelním počítači patřičně distribuována podle toho,

který proces je vyžaduje. Potom také paměťová náročnost řešeného problému může být výrazně nižší. Protože časová náročnost vstupně/výstupních operací (čtení/zápis dat) je důležitá při implementaci každého algoritmu a následné optimalizaci výsledného kódu, je příznivé, že každý proces pracuje pouze s daty, která nezbytně potřebuje a počet vstupně/výstupních operací je co nejnižší. Další výhodou je pak rychlejší operace čtení záznamů z datového souboru při jeho průchodu směrem dopředu i zpět. Tato vlastnost nabyde na významu zejména v případě práce s velkými soubory.



Obrázek 35: Uspořádání prvků horní trojúhelníkové části submatic A_{ii} při separaci složek posunutí.



Obrázek 36: Uspořádání prvků submatic A_{ij}, $i \neq j$ při separaci složek posunutí.

Submatice A_{ii}, $i = 1, 2, 3$ jsou symetrické a proto je uložena pouze horní trojúhelníková část každé matice. V každém řádku horní trojúhelníkové části A_{ii} je 14 pozic definovaných pravidelnou diskretizační šablonou, v nichž budou všechny nenulové prvky. Pravidelná diskretizační šablona také definuje 27 pozic se všemi nenulovými prvky každého řádku submatic A_{ij}, $i, j = 1, 2, 3$, $i \leq j$. Další vlastnosti datových struktur související s uložením jejich prvků (doplnění posledních řádků submatic nulami a prodloužení vektorů) jsou stejné jako v případě datových struktur v původním, kompaktním tvaru.

Schéma algoritmu

Při použití separace složek posunutí odpovídajících datových struktur úlohy můžeme soustavu lineárních algebraických rovnic

$$Au = f$$

vznikající řešením 3D úloh pružnosti zapsat ve tvaru

$$\begin{aligned} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 &= f_1 \ , \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 &= f_2 \ , \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 &= f_3 \ , \end{aligned}$$

kde u_1, u_2, u_3 jsou složky posunutí po řadě ve směrech $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ a f_1, f_2, f_3 jsou složky pravé strany, která je rozdělená stejným způsobem. Tedy $u = (u_1, u_2, u_3)$ a $f = (f_1, f_2, f_3)$.

Pro řešení soustavy využijeme tři samostatné, identické, vzájemně komunikující procesy (dělníky), jenž se liší pouze zpracovávanými daty v závislosti na složce posunutí, kterou řeší. Pro spouštění celého programu, vzájemnou synchronizaci dělníků, obsluhu uživatelských vstupů a zajištění výpisů informací využijeme další, řídící proces. Vzhledem k objemu vykonané práce jsou dělníci daleko více vytížení než řídící proces, proto jej lze spouštět spolu s některým dělníkem na jednom procesoru nebo samostatně. Pro úplnost poznamenejme, že se podle počtu úloh jedná o hrubou paralelizaci, z hlediska vytváření/rušení úloh během výpočtu o statickou paralelizaci a z pohledu rozdělení dat o datovou dekompozici problému.

Jak můžeme vidět z obrázku č. 37, algoritmus paralelní verze sdružených gradientů je podobný verzi sekvenční, kdy výpočet probíhá rovněž ve dvou fázích vzájemně oddělených ukončovacím kritériem. V levém sloupci je algoritmus řídícího procesu, v pravém algoritmus každého dělníka. Vzájemné rozlišení dělníků je realizováno indexem k. Čárkovanými šipkami uprostřed je znázorněna komunikace mezi řídícím procesem a dělníky, vpravo pak vzájemná komunikace mezi k-tým a ostatními dělníky odlišenými indexem i.

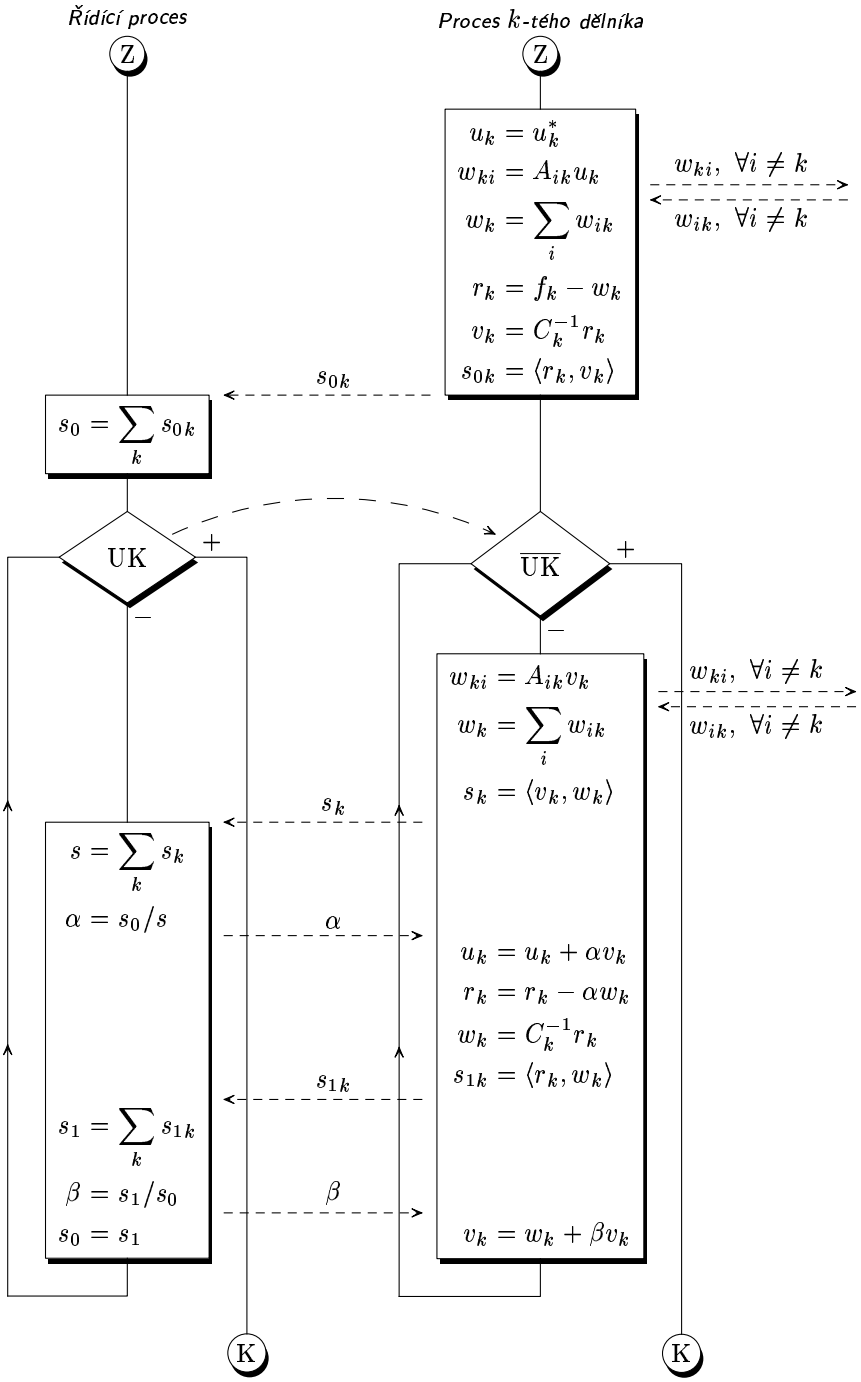
Ukončení iterací

Ukončovací kritérium řídícího procesu UK je stejné jako v případě sekvenční verze, tedy

$$\frac{\|r\|}{\|f\|} = \sqrt{\frac{\langle r, r\rangle}{\langle f, f\rangle}} \leq \varepsilon \ .$$

Protože však řídící proces neuchovává kopie residua r a pravé strany f, které jsou navíc rozděleny a každý dělník pracuje pouze s určitými částmi těchto vektorů, pošle mu na začátku přípravné fáze každý dělník dílčí skalární součin příslušné části pravé strany

$$\|f_k\|^2 = \langle f_k, f_k\rangle$$



Obrázek 37: Paralelní verze algoritmu metody sdružených gradientů s předpodmíněním založená na separaci složek posunutí (`DiD`). Čárkovaně uprostřed je znázorněna komunikace řídícího procesu s dělníky, vpravo pak komunikace mezi dělníky. Indexy $k, i = 1, 2, 3$.

a v každé iteraci skalární součin příslušné části residua

$$\|r_k\|^2 = \langle r_k, r_k \rangle \ .$$

Pro řídící proces pak není obtížné po jednoduché numerické úpravě vyhodnotit ukončovací podmínku. Po tomto vyhodnocení pošle řídící proces dělníkům zprávu o tom, zda mají pokračovat ve výpočtu další iterací nebo uložit řešení do datových souborů a ukončit svoji činnost. Příjem této zprávy dělníky můžeme chápat jako jejich ukončovací kritérium $\overline{\mathrm{UK}}$, které je závislé na ukončovacím kritériu řídícího procesu UK. Tato závislost je v obrázku č. 37 zdůrazněna čárkovanou čárou s hrubším vzorem.

Operace násobení matice × vektor

Každá iterace metody sdružených gradientů vyžaduje vykonání operace násobení matice × vektor

$$w = Av \ ,$$

kterou můžeme přepsat do tvaru

$$\begin{aligned} w_1 &= A_{11}v_1 + A_{12}v_2 + A_{13}v_3 \ , \\ w_2 &= A_{21}v_1 + A_{22}v_2 + A_{23}v_3 \ , \\ w_3 &= A_{31}v_1 + A_{32}v_2 + A_{33}v_3 \ , \end{aligned}$$

což znamená, že při této operaci spolu musí dělníci komunikovat. Aby k-tý dělník nemusel uchovávat celý vektor $v = (v_1, v_2, v_3)$, který se v každé iteraci aktualizuje, bude výhodnější, když bude udržovat pouze jeho příslušnou část v_k. Potom ale musí navíc spočíst

$$w_{ki} = A_{ik}v_k \ , \qquad i = 1, 2, 3 \quad i \neq k$$

za ostatní dělníky, vyměnit s nimi dvojici vektorů w_{ki} za dva příspěvky w_{ik} k vektoru w_{kk} a provést patřičný součet w_k.

Operace předpodmínění

Předpodmínění je přirozené založit na stejné metodě rozkladu prostoru, jenž byla využita pro paralelizaci algoritmu metody sdružených gradientů. Předpodmínění separací složek posunutí `DiD` bylo popsáno v oddíle 4.2.1.

4.3.2 Algoritmus pro rozklad oblasti s překrytím `DD`

Další technikou, kterou můžeme využít pro paralelizaci metody sdružených gradientů s předpodmíněním, je rozklad studované oblasti na překrývající se podoblasti. Takový algoritmus, který bude popsán v tomto oddíle, budeme označovat `DD` podle využité metody rozložení prostoru.

Popis datových struktur

Algoritmus pracuje s daty, které odpovídají rozkladu oblasti na m podoblastí s překrytím podle obrázků č. 31, 32 v oddíle 4.2.2. Ze schématu rozkladu oblasti vyplývá, že uspořádání prvků submatic A_i, $i = 1, \ldots, m$, je stejné jako tomu bylo v případě matice A v původním, kompaktním tvaru, který je zachycen na obrázku č. 27 v oddíle 4.1.1. Proto uložení submatic A_i do datových souborů a práce s nimi bude probíhat podle stejných pravidel.

Potom celkový objem výpočtů může být distribuován do m paralelních procesů, jenž odpovídají jednotlivým podoblastem. Navíc pro zefektivnění celé metody lze využít řešení na jednotlivých podoblastech.

Samotný způsob rozložení oblasti můžeme libovolně zobecnit. Uvedený postup je však výhodný pro implementaci tím, že nezbytná vzájemná komunikace paralelních procesů probíhá vždy pouze mezi dvěma procesy, které odpovídají sousedním podoblastem.

Schéma algoritmu

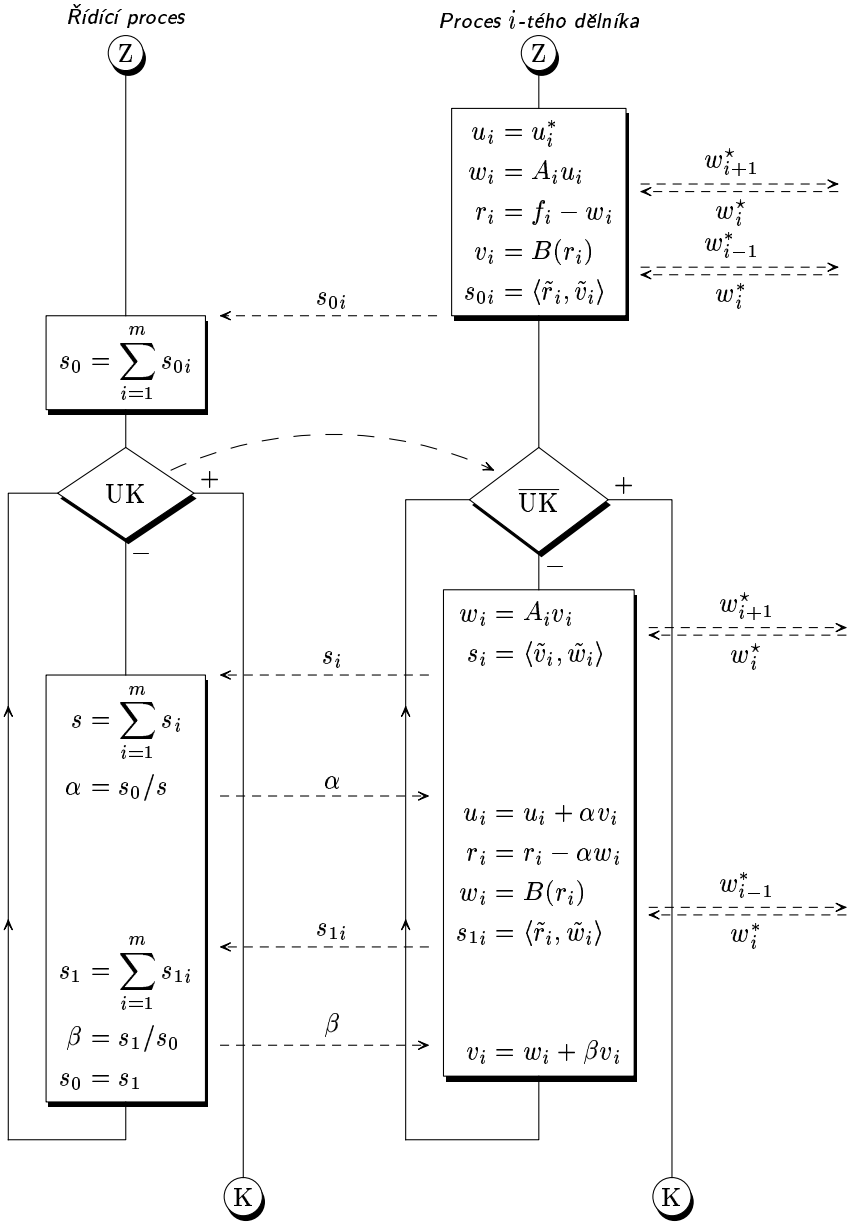
Předpokládejme popsané rozložení oblasti Ω na překrývající se podoblasti Ω_i a existenci odpovídajících matic tuhosti A_i a vektorů zatížení f_i. Dále předpokládejme, že v maticích A_i jsou zohledněny hlavní okrajové podmínky odpovídající $\partial\Omega_i \cap \partial\Omega$. Na zbývající části $\partial\Omega_i$ je Dirichletova okrajová podmínka.

K řešení celkového lineárního systému

$$Au = f$$

použijeme algoritmus metody sdružených gradientů s předpodmíněním ve tvaru podle obrázku č. 38. Pro výpočet využijeme m samostatných, funkčně identických, vzájemně komunikujících procesů (dělníků), které se liší pouze zpracovávanými daty, jenž odpovídají jednotlivým podoblastem Ω_i. Běhy dělníků řídí další, výpočetně nenáročný proces, jehož význam je stejný, jako tomu bylo v případě paralelního algoritmu `DiD`. Během výpočtu nedochází k vytváření/rušení procesů, jedná se o paralelizaci statickou. Celkový počet dělníků m je však možné měnit například s ohledem na velikost řešeného problému nebo počet dostupných procesorů paralelního počítače, proto se zároveň jedná o paralelizaci jemnou.

Ze schématu paralelního algoritmu `DD` je patrná jeho podoba s paralelním algoritmem `DiD`. Výpočet probíhá ve dvou fázích oddělených ukončovacím kritériem. V levém sloupci je algoritmus řídícího procesu, v pravém algoritmus každého dělníka. Čárkovanými šipkami uprostřed je znázorněna komunikace mezi řídícím procesem a dělníky, vpravo pak vzájemná komunikace mezi i-tým dělníkem a jeho sousedy s indexy $i+1$ a $i-1$.



Obrázek 38: Paralelní verze algoritmu metody sdružených gradientů s předpodmíněním založená na rozložení oblasti (DD). Čárkovanou čárou uprostřed je znázorněna komunikace řídícího procesu s dělníky, vpravo pak komunikace mezi dělníky. Index $i = 1, \ldots, m$, kde m je celkový počet podoblastí.

Ukončení iterací

O ukončení iteračního procesu rozhoduje řídící proces na základě splnění ukončovacího kritéria UK ve tvaru

$$\frac{\|r\|}{\|f\|} = \sqrt{\frac{\langle r, r\rangle}{\langle f, f\rangle}} \leq \varepsilon \ .$$

Stejně jako v případě algoritmu `DiD` řídící proces neuchovává kopie residua r a pravé strany f, dílčí skalární součiny příslušných částí pravé strany

$$\|\tilde{f}_i\|^2 = \ \langle \tilde{f}_i, \tilde{f}_i\rangle$$

a residua

$$\|\tilde{r}_i\|^2 = \ \langle \tilde{r}_i, \tilde{r}_i\rangle$$

mu pošle každý dělník na začátku přípravné fáze, resp. v každé iteraci. Je přirozené, že uvedené dílčí skalární součiny počítá i-tý dělník pouze z nepřekrývajících se částí vektorů pravé strany a residua, tj. z vektorů $\tilde{f}_i$, $\tilde{r}_i$. Po vyhodnocení ukončovacího kritéria UK, jehož upravený tvar je

$$\sqrt{\frac{\sum_{i=1}^{m}\langle \tilde{r}_i, \tilde{r}_i\rangle}{\sum_{i=1}^{m}\langle \tilde{f}_i, \tilde{f}_i\rangle}} \leq \varepsilon \ ,$$

pošle řídící proces všem dělníkům zprávu o tom, zda mají pokračovat ve výpočtu další iterací nebo uložit řešení do datových souborů a ukončit svoji činnost. Příjem této zprávy dělníky je současně jejich ukončovací podmínkou $\overline{\mathrm{UK}}$. Závislost ukončovací podmínky $\overline{\mathrm{UK}}$ na podmínce UK zdůrazňuje v obrázku č. 38 čárkovaná šipka s hrubším vzorem.

Operace násobení matice × vektor

V každé iteraci sdružených gradientů se globální operace násobení matice × vektor

$$w = Av$$

realizuje pomocí výpočtů

$$w_i = A_i v_i \ , \qquad i = 1, \ldots, m \ ,$$

a nezbytné komunikace mezi dělníky. Přirozená je implementace, kdy i-tý dělník počítá dílčí operaci submatice krát vektor pouze na nepřekrývající se části podoblasti $\tilde{\Omega}_i$,

$$\tilde{w}_i = \tilde{A}_i \tilde{v}_i \ .$$

Současně vytváří vektor $w^{\star}_{i+1}$ příspěvků k vektoru w_{i+1} ze součinů prvků vektoru $\tilde{v}_i$ a těch nenulových prvků submatice A_i, které nepatří do $\tilde{A}_i \cup A_{i+1}$. Vektor $w^{\star}_{i+1}$

obsahuje celkově 3*NX*NY+3*NX+6 prvků, jeho velikost je závislá na parametrech diskretizační sítě a odpovídá poloviční šířce pásu matice A. Po sestavení $w^{\star}_{i+1}$ jej i-tý dělník posílá dělníkovi i+1, který jej přičítá na začátek vektoru w_{i+1}.

Operace předpodmínění

Metodu rozkladu prostoru, aplikovanou pro paralelizaci algoritmu metody sdružených gradientů, je přirozené zároveň využít ke konstrukci předpodmínění. Odpovídající jednoúrovňové a dvouúrovňové předpodmiňovače, založené na rozkladu oblasti, byly popsány v oddílech 4.2.2 a 4.2.3.

Aplikace hrubé sítě

Do paralelního algoritmu DD je přirozené implementovat řešení na hrubé síti jako další samostatný proces, který spolupracuje s ostatními dělníky při výpočtu předpodmínění. Dvouúrovňový předpodmiňovač aditivního typu, pracující s agregovanou hrubou sítí, může být realizován tak, že nejprve dělníci pracující na podoblastech posílají dělníkovi na hrubé síti agregované vektory $\tilde{r}_i$. Ten z nich sestaví agregované residum r a následně řeší operaci s maticí A_c^{-1} vnitřními iteracemi metody sdružených gradientů, zatímco ostatní dělnící vykonávají operaci předpodmínění na podoblastech stejně jako tomu bylo u jednoúrovňového předpodmiňovače. Na závěr dělník na hrubé síti posílá části patřičně rozděleného výsledného vektoru svého výpočtu ostatním dělníkům, kteří jejich přičtením aktualizují své lokální výsledky w_i. Popsaným způsobem pracuje řešení na hrubé síti paralelně s výpočty na jednotlivých podoblastech. Celkové množství komunikace dělníků se oproti jednoúrovňovému předpodmiňovači zvýší o přenos 6*NX_A*NY_A*NZ_A reálných čísel, kde NX_A, NY_A, NZ_A jsou počty uzlů agregované hrubé sítě ve směrech souřadnicových os $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$.

Poznamenejme, že pokud by dělníci na podoblastech posílali celé, neagregované vektory $\tilde{r}_i$, potom by zvýšení časové náročnosti většího množství komunikace dělníků během předpodmínění mohlo úplně znehodnotit výhody vyplývající z nižšího počtu iterací celé metody, jak bude potvrzeno výsledky testu v oddíle 6.5. Tyto výsledky současně ukazují, že k docílení maximální efektivity celé metody je třeba věnovat zvláštní pozornost vyvážení zátěže procesorů. To znamená, stanovit velikost hrubé sítě tak, aby během každé iterace korespondoval celkový objem výpočtů dělníka na hrubé síti s celkovými objemy výpočtů dělníků na jednotlivých podoblastech.

U hybridního typu dvouúrovňového předpodmiňovače není možné celou operaci předpodmínění paralelizovat, řešení na hrubé síti musí předcházet výpočtům na jednotlivých podoblastech. Samotná implementace i komunikační náročnost hybridního předpodmiňovače se neliší od aditivního typu. Jediný rozdíl je v tom, že dělníci na podoblastech musí s výpočty předpodmínění čekat do okamžiku, než dělník na hrubé síti dokončí operaci s maticí A_c^{-1} a pošle jim vektor, který po vynásobení maticí tuhosti stanoví opravu residua. Dále pokračuje operace předpodmínění výpočty na podoblastech jako u jednoúrovňového předpodmiňovače.

5 Realizace programů

Záměr získat efektivní řešič soustavy lineárních rovnic vedl k programové realizaci sekvenčních a paralelních algoritmů popsaných v kapitole 4. V programovacím jazyce Fortran 77 byly vytvořeny odpovídající přenositelné programy, které jsou součástí knihovny řešičů `ELPAR`, resp. systému `PortaGEM`. Jeho archiv je s příslušným oprávněním dostupný na adrese:

`ftp://fany.ugn.cas.cz/u/gem/portagem`

Snaha o dosažení maximální efektivity výsledných kódů zapříčinila kontinuální zdokonalování programů. Další verze programů vznikají optimalizací kódů verzí předešlých, přidáváním nových funkčních možností a odstraňováním chyb. Tento přirozený a dlouhodobý vývoj se netýká pouze samotných řešičů, ale také souvisejících programů a pomocných utilit. Vzhledem k převážně experimentálnímu charakteru programů stručně popíšeme v dalším textu pouze základní vlastnosti aktuálních verzí řešičů.

5.1 Sekvenční řešič `SESOLO`

Sekvenční řešič `SESOLO` verze 1.00a (7. 11. 2000) pracuje podle sekvenčního algoritmu metody sdružených gradientů s předpodmíněním (na obrázku č. 25). Řešič zpracovává data fyzicky uložená v původním, kompaktním tvaru. Vektory udržuje v operační paměti, zatímco prvky matice tuhosti čte v každé iteraci z disku. Čtení probíhá po jednotlivých záznamech, které odpovídají vždy trojici řádků matice tuhosti. Podle způsobu práce s maticí tuhosti proto označujeme řešič jako diskově orientovaný, jenž má minimální paměťové nároky. Řešič může provést výpočet bez předpodmínění nebo využít předpodmiňovač založený na neúplné faktorizaci.

Umístění zdrojových textů v `PortaGEMu`: `/solver.js/sesolo-100a`

5.2 Sekvenční řešič `SESOL`

Sekvenční řešič `SESOL` verze 1.20a (24. 7. 2001) pracuje podle sekvenčního algoritmu metody sdružených gradientů s předpodmíněním (na obrázku č. 25). Řešič zpracovává data fyzicky uložená v původním, kompaktním tvaru. Po svém startu řešič nejprve načte data (včetně matice tuhosti) z disku do paměti, kde je následně dále zpracovává iteračním procesem. Podle způsobu práce s maticí tuhosti proto označujeme řešič jako paměťově orientovaný.

Řešič může provést výpočet bez předpodmínění, využít diagonální předpodmínění, neúplnou faktorizaci nebo vnitřní iterace. Navíc může výpočet doplnit o dodatečnou ortogonalizaci nového směru postupu k jednomu poslednímu směru postupu (při hledání řešení).

Umístění zdrojových textů v `PortaGEMu`: `/solver.js/sesol-120a`

5.3 Paralelní řešič ITERA

Paralelní řešič ITERA verze 4.10a (25. 7. 2001) pracuje podle paralelního algoritmu metody sdružených gradientů s předpodmíněním DiD (na obrázku č. 37), který je založen na separaci složek posunutí datových struktur úlohy. Řešič zpracovává všechna data v paměti, kam je bezprostředně po svém startu načte z disku.

Řešič může provést výpočet bez předpodmínění, využít pro předpodmínění neúplnou faktorizaci nebo vnitřní iterace. Navíc může výpočet doplnit o dodatečnou ortogonalizaci nového směru postupu k jednomu poslednímu směru postupu (při hledání řešení) nebo o zapojení projekce pro řešení lineárního systému se singulární maticí soustavy.

Součástí řešiče jsou rozhraní pro PVM a MPI, které umožňují provozovat řešič v uvedených typech systémů předávání zpráv. Samotný řešič doplňují další, pomocné utility. Program DSPLIT slouží ke konverzi datových struktur lineárního systému z původního, kompaktního tvaru do tvaru se separovanými složkami posunutí. Program SMERGE umí sestavit řešení lineárního systému z dílčích částí, které odpovídají separovaným složkám posunutí, zpátky do kompaktního tvaru.

Umístění zdrojových textů v PortaGEMu: `/parsol.js/itera-410a`

5.4 Paralelní řešič ISOL

Paralelní řešič ISOL verze 1.45a (9. 7. 2002) pracuje podle paralelního algoritmu metody sdružených gradientů s předpodmíněním DD (na obrázku č. 38), který je založen na rozkladu studované oblasti na překrývající se podoblasti. Dekompozici dat odpovídá i jejich uložení na disku, odkud je řešič bezprostředně po svém spuštění načte do paměti, kde je dále zpracovává.

Řešič může výpočet provést bez předpodmínění, využít diagonální předpodmínění nebo neúplnou faktorizaci. Výpočet ovlivňuje také volba počtu podoblastí, na které se rozdělí původní oblast, a šířka jejich překrytí. Dále je možné při výpočtu využít explicitní nebo agregovanou hrubou síť, přičemž lze měnit hustotu hrubé sítě a typ dvouúrovňového Schwarzova předpodmiňovače (aditivní), dodatečnou ortogonalizaci nového směru postupu k jednomu poslednímu směru postupu (při hledání řešení) nebo projekci pro řešení lineárního systému se singulární maticí soustavy.

Aby mohl být řešič provozován pod PVM nebo MPI, jsou jeho součástí rozhraní pro uvedené systémy předávání zpráv. Samotný řešič doplňují drobné pomocné utility. Program DCONV umožňuje rozklad datových struktur lineárního systému, který odpovídá dekompozici studované oblasti na překrývající se podoblasti. Program SREST sestavuje řešení z dílčích částí do kompaktního tvaru. Program MAGR slouží ke generování agregované hrubé sítě.

Umístění zdrojových textů v PortaGEMu: `/parsol.js/isol-145a`

5.5 Optimalizace řešičů

Postupné zdokonalování řešičů zapříčinilo řadu úprav v jejich programových kódech. Pomineme-li opravy chyb a doplňování různých algoritmických variant, týkala se převážná část všech změn následujících oblastí:

- Přechod od diskově k paměťově orientovaným programům. Hlavní motivací byla snaha odstranit vliv diskového subsystému počítače na celkový čas řešení, který jsme považovali ze prvotní kritérium pro posuzování rychlosti/kvality řešiče.

- Optimalizace komunikace. Jejím cílem bylo redukovat celkový objem všech komunikací i počet nezbytných synchronizačních bodů v paralelních programech a využít alespoň částečného vzájemného překrytí komunikací tak, aby nedocházelo k nepříznivým časovým prodlevám během interakcí paralelních procesů.

- Vyvažování zátěže jednotlivých procesorů. V případě paralelního řešiče `DD` bylo nutné navrhnout algoritmus rozkladu původní oblasti na zadaný počet částí tak, aby výsledné překrývající se (nebo případně nepřekrývající se) podoblasti měly stejnou nebo podobnou velikost.

- Experimentální hledání a nastavení hodnot papametrů ovlivňujících běhy programů, například ε^*. Připomeneme, že se jedná o hodnotu relativní přesnosti řešení podúloh vnitřními iteracemi v případech předpodmínění `DiD-IE` nebo při aplikaci hrubé sítě u paralelního řešiče `DD`.

- Přenositelnost kódů. Záměr disponovat univerzálními řešiči, které by bylo možné provozovat na různých platformách paralelních počítačů, zapříčinil úpravy částí kódů řešičů, jenž byly závislé na cílové architektuře nebo využívaly různých rozšíření konkrétních překladačů. Hlavně šlo o:

  - striktní dodržování normy pro programovací jazyk Fortran 77 bez využívání různých rozšíření, jež překladač na cílové architektuře dovoluje,
  - vyloučení nebo minimalizování počtu implementačně závislých funkcí, například čtení systémového času, a funkcí obsažených v nestandardních knihovnách.

6 Testování řešičů

Abychom mohli zhodnotit realizované řešiče, vykonali jsme řadu testovacích výpočtů na modelových úlohách z geomechaniky, které byly popsány v oddíle 3.4. Všechny testy byly provedeny na paralelních počítačích, jejichž charakteristiky byly uvedeny v oddíle 2.1.6.

Během dlouhodobého vývoje se měnily nejen samotné řešiče a pomocné utility, ale také testovací úlohy, konfigurace jednotlivých počítačů i přepínače ovlivňující běh a chování programů. V konečném důsledku všechny tyto změny zapříčinily vykonání velkého počtu různých testů, které byly zaměřeny na: odstranění chyb programů, optimalizaci kódů řešičů a posouzení jejich chování, dosažení maximální rychlosti a efektivity řešičů, doplnění programů o další funkční možnosti pro řešení širšího okruhu problémů.

V této kapitole se zaměříme zejména na nejvyspělejší verze řešičů a jejich testování na platformách SUN-1 a LINUX-4, neboť tyto testy byly nejkomplexnější a s ohledem na dostatečný počet procesorů paralelních počítačů i stabilitu a neměnnost jejich systémového prostředí proběhly za nejpříznivějších podmínek.

Pro komplexní zhodnocení testů je důležité uvádět nejen konfigurace počítačů, ale také popis souvisejících programových prostředků. Proto pokud nebude v dalším textu uvedeno jinak[9], byly testy prováděny za následujících podmínek:

- Na všech architekturách byly zdrojové texty řešičů přeloženy do spustitelného tvaru s maximální úrovní optimalizace, kterou cílová architektura dovolovala, zpravidla `-O3`.
- Relativní přesnost řešení ε byla stanovena na hodnotu 10^{-4}. V případě proměnného předpodmínění nebo výpočtů na hrubé síti byly vnitřní iterace omezeny relativní přesnosti řešení $\varepsilon^* \leq 10^{-1}$.
- Časy řešení, uváděné ve výsledcích testů, znamenají celkovou dobu výpočtu řešení, která nezahrnuje čas potřebný pro čtení dat ze souborů na disku do paměti a čas potřebný pro zápis výsledného řešení do datových souborů. Takto naměřené hodnoty by neměly být ovlivněny například umístěním datových souborů (lokální disky, NFS souborové systémy a podobně). Měření času bylo realizováno samotnými řešiči.
- Výpočty s explicitní hrubou sítí využívaly hybridní předpodmiňovač, při všech ostatních výpočtech se uplatnil předpodmiňovač aditivní.
- V tabulkách s výsledky testů je použito pro sekvenční/paralelní řešič označení S/P. U všech řešičů jsou uvedeny jejich charakteristické vlastnosti, například typ předpodmínění, typ paralelního algoritmu a typ hrubé sítě.

[9] První verze řešičů byly z hlediska práce s maticí tuhosti diskově orientované. Výpočty s těmito řešiči byly prováděny na platformě IBM SP a byly omezeny stanovením relativní přesnosti řešení ε na hodnotu 10^{-3}.

6.1 První srovnání výkonů řešičů

Cílem prvního testu bylo získání prvního srovnání výkonů všech řešičů a zhodnocení možností paralelizace z pohledu řešení dané úlohy určitou třídou algoritmů založených na metodě sdružených gradientů s předpodmíněním.

Pro testy jsme vybrali úlohu FOOT 40 (192 000 rovnic). Jedná se o čistě akademický, původní model zatížení podloží základu. Všechny výpočty byly provedeny na platformě IBM SP. Řešiče byly z hlediska práce s maticí tuhosti diskově orientované, neboť neudržovaly prvky matice tuhosti trvale v operační paměti, ale četly je v každé iteraci ze souboru na disku. Iterační procesy byly omezeny dosažením hodnoty relativní přesnosti řešení $\varepsilon = 10^{-3}$.

FOOT 40						
Řešič		Počet	Počet	Čas řešení	Zrychlení	Účinnost
Typ	Vlastnosti	procesorů	iterací	T [s]	S	E
S	IF	1	49	420		
P	DiD, IF	3	49	223	1.88	0.63
P	DiD, IE	3	08	202	2.08	0.69
P	DD, CG	5	17	90	**4.67**	**0.93**

Tabulka 4: Efektivita paralelizace sdružených gradientů, IBM SP: FOOT 40.

Výsledky testu jsou shrnuty v tabulce č. 4. Paralelní řešič DiD s neúplnou faktorizací byl při výpočtu na třech procesorech téměř 2× rychlejší než sekvenční řešič se stejným typem předpodmiňovače.

Hranici dvojnásobného zrychlení na stejném počtu procesorů překonal paralelní řešič DiD s předpodmiňovačem realizovaným iteracemi vnitřní metody sdružených gradientů. Program dosáhl kratšího času řešení, a tím také větší hodnoty zrychlení, především díky nízkému počtu vnějších iterací. Většinu celkového objemu výpočetní práce potřebné pro nalezení řešení vykonaly vnitřní iterace, které nevyžadují vzájemnou komunikaci dělníků a pracují pouze s diagonálními částmi matice tuhosti.

Nejkratšího času řešení dosáhl paralelní řešič DD. Původní oblast byla rovnoměrně rozložena na čtyři podoblasti, které se překrývaly o jednu vrstvu uzlů diskretizační sítě (OF=1). Při výpočtu byla v hybridním Schwarzově předpodmiňovači využita explicitní hrubá síť FOOT 12 (5184 rovnic). Program spuštěný na pěti procesorech provedl výpočet přibližně 4.7× rychleji než sekvenční řešič.

Z výsledků testu celkově vyplývá, že techniky paralelizace metody sdružených gradientů založené na separaci složek posunutí a rozkladu oblasti s překrytím mohou být velmi efektivní, což potvrzuje účinnost, která při posledním výpočtu dosáhla vysoké hodnoty 0.93.

6.2 Rozklad oblasti s překrytím, hrubá síť

Ve druhém testu jsme zaměřili pozornost na chování paralelního algoritmu DD v závislosti na počtu podoblastí a rozsahu jejich překrytí OF. Současně jsme zkoumali možnost využití explicitní hrubé sítě při výpočtu.

Pro testování jsme vybrali úlohu FOOT 40, pro vytvoření explicitní hrubé sítě úlohy FOOT 04 a FOOT 12. Všechny výpočty byly provedeny na platformě IBM SP, přičemž byly ukončeny dosažením relativní přesnosti řešení $\varepsilon = 10^{-3}$. Stejně jako v případě prvního testu, byly řešiče diskově orientované. Podúlohy na podoblastech byly řešeny neúplnou faktorizací IF a podúloha na explicitní hrubé síti vnitřními iteracemi IE.

Počet	OF = 0		OF = 1		OF = 2		OF = 3		OF = 4	
podoblastí	It	T [s]	It	T [s]	It	T [s]	It	T [s]	It	T [s]
FOOT 40										
2	88	793	69	638	64	611	67	660	67	666
3	126	778	93	581	87	569	85	573	82	580
4	133	603	105	505	100	508	98	520	95	527
FOOT 40 + FOOT 04										
2	31	293	24	239	24	250	25	260	26	277
3	40	260	31	204	29	200	29	209	29	220
4	43	204	34	172	33	179	33	189	33	192
FOOT 40 + FOOT 12										
2	17	171	14	186	13	145	14	157	15	170
3	21	158	16	126	15	121	15	122	15	123
4	21	108	**17**	**90**	17	99	17	102	17	103

Tabulka 5: Rozklad oblasti na překrývající se podoblasti. Vzájemný vztah počtu podoblastí, rozsahu jejich překrytí OF a efektivity Schwarzova předpodmiňovače (jednoúrovňový aditivní / dvouúrovňový hybridní), IBM SP: FOOT.

Výsledky testu jsou shrnuty v tabulce č. 5, která zachycuje počet iterací It a celkový čas řešení T dané úlohy s ohledem na počet podoblastí a rozsah jejich překrytí OF. V horní části tabulky jsou zaznamenány výsledky výpočtů bez hrubé sítě, v prostřední části s hrubou sítí o 4×4×4 uzlech (192 rovnic) a v dolní části s hrubou sítí o 12×12×12 uzlech (5184 rovnic).

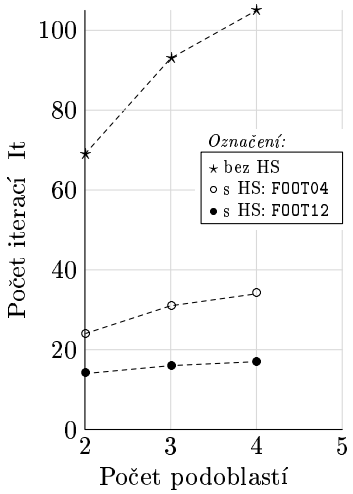
Z tabulky jsou patrné některé charakteristické vlastnosti algoritmů založených na Schwarzových metodách:

- Celkový počet iterací It potřebných pro nalezení řešení roste s počtem podoblastí, se zvětšujícím se rozsahem překrytí podoblastí OF naopak klesá. Rozsah překrytí podoblastí současně ovlivňuje také nárůst množství výpočetní práce na podoblastech. V praxi se obvykle aplikuje minimální rozsah překrytí podoblastí, OF=1.

- Využitím hrubé sítě při výpočtu lze výrazně snížit celkový počet iterací It, který klesá s hustotou hrubé sítě. Situaci zachycuje obrázek č. 39. Chování algoritmu DD se změní, neboť se sníží rozdíly mezi počty iterací pro různé počty podoblastí, podobně jako rozdíly v počtech iterací pro různé rozsahy překrytí podoblastí.

Poznamenejme, že volba hrubé sítě z hlediska její hustoty není triviální. Z chování paralelního algoritmu DD vyplývá, že čím je hrubá síť hustší, tím je celkový počet iterací menší. Zároveň však výpočet na hrubé síti trvá delší dobu. Je nutné si uvědomit, že při aplikovaném dvouúrovňovém hybridním typu předpodmínění výpočet na explicitní hrubé síti neběží s výpočty na podoblastech paralelně, ale sekvenčně. Doba jeho provádění tedy přímo ovlivňuje dobu provádění jedné iterace metody sdružených gradientů a tím i celkový čas řešení lineárního systému.

Rovněž v případě aplikace dvouúrovňového aditivního předpodmínění je pro dosažení maximální účinnosti paralelního řešiče DD důležité vhodně stanovit hustotu hrubé sítě tak, aby byly vyváženy výpočty podúloh na podoblastech neúplnou faktorizací a na hrubé síti vnitřními iteracemi.



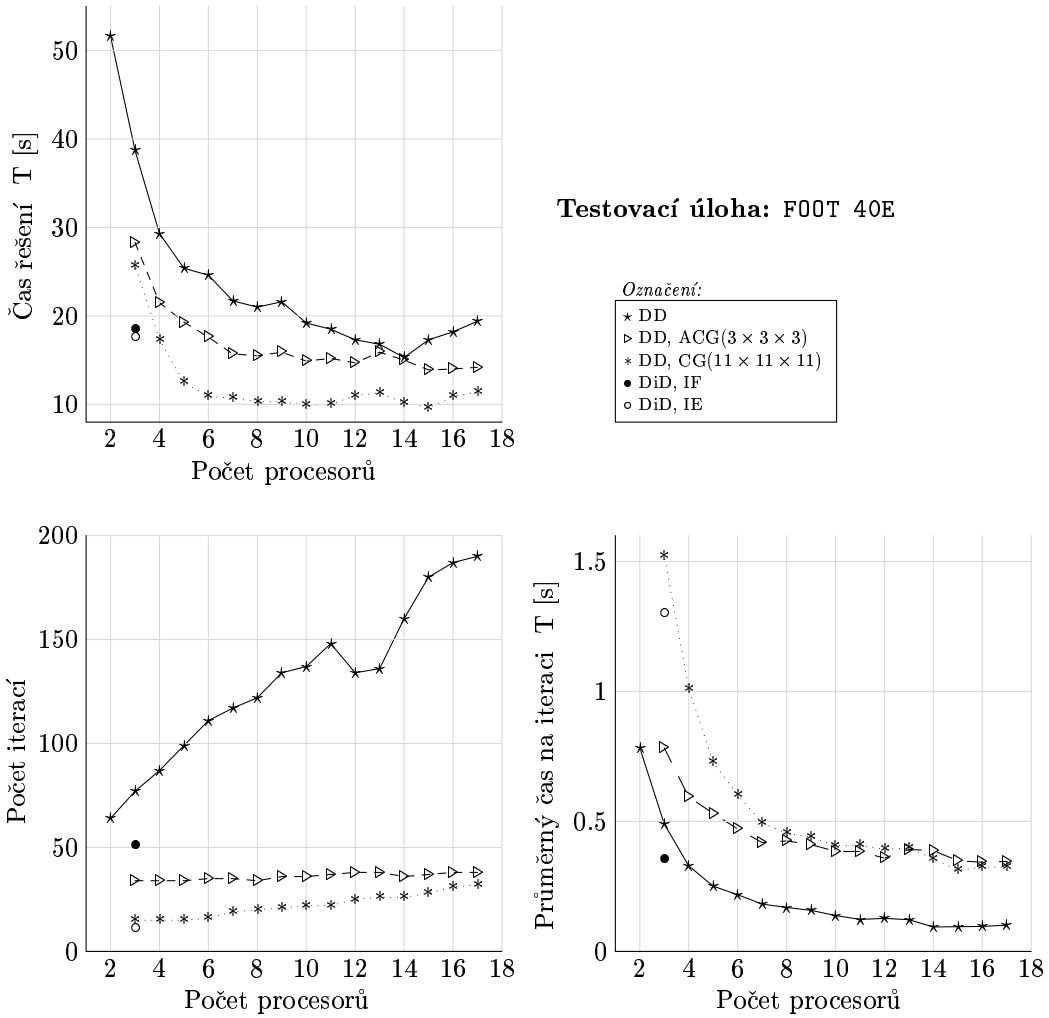
Obrázek 39: Rozklad oblasti s překrytím, OF=1. Výpočet bez hrubé sítě a s méně i více hustou explicitní hrubou sítí.

6.3 Detailní srovnání výkonů řešičů

Nejrozsáhlejší skupina testů byla zaměřena na detailní srovnání výkonů paralelních řešičů, zejména z hlediska doby provádění výpočtu na daném počtu procesorů a počtu iterací potřebných pro nalezení řešení.

Testování proběhlo na platformě SUN-1. Pro výpočty jsme využili testovací úlohy FOOT 40 E a FOOT 60 E, na nichž jsme zkoušeli paměťově orientované verze všech řešičů. Každou úlohu jsme řešili sekvenčním řešičem s neúplnou faktorizací, paralelním řešičem DiD s neúplnou faktorizací (IF) a s vnitřními iteracemi (IE) a paralelním řešičem DD bez využití hrubé sítě a s explicitní (CG) i agregovanou (ACG) hrubou sítí. Pro vytvoření explicitní hrubé sítě jsme použili úlohu FOOT 10 E. Agregacemi tvořenými třemi uzly v každém směru (AF=3) jsme odvodili agregovanou hrubou síť. V případě explicitní hrubé sítě se jednalo o dvouúrovňový Schwarzův předpodmiňovač hybridní, se kterým při testování paralelní řešič dosahoval kratších časů provádění výpočtu i menšího počtu iterací než s předpodmiňovačem aditivním. V případě agregované hrubé sítě se uplatnil předpodmiňovač aditivní, neboť s hybridním předpodmiňovačem řešič zpravidla nekonvergoval a pro správnou funkci by vyžadoval úpravu pro svou stabilizaci.

Na obrázku č. 40 jsou výsledky testů na úloze FOOT 40 E (206 763 rovnic). Paralelní řešič DiD s neúplnou faktorizací dosáhl na symetrickém multiprocesoru SUN-1 se sdílenou pamětí, která umožňovala mimořádně rychlé předávání zpráv mezi paralelními procesy, přibližně stejného času řešení jako s vnitřními iteracemi, ačkoliv potřeboval téměř 5× více iterací. Na paralelním počítači s pomalejšími komunikačními linkami proto efektivita předpodmínění vnitřními iteracemi bude vyšší než efektivita předpodmínění neúplnou faktorizací.

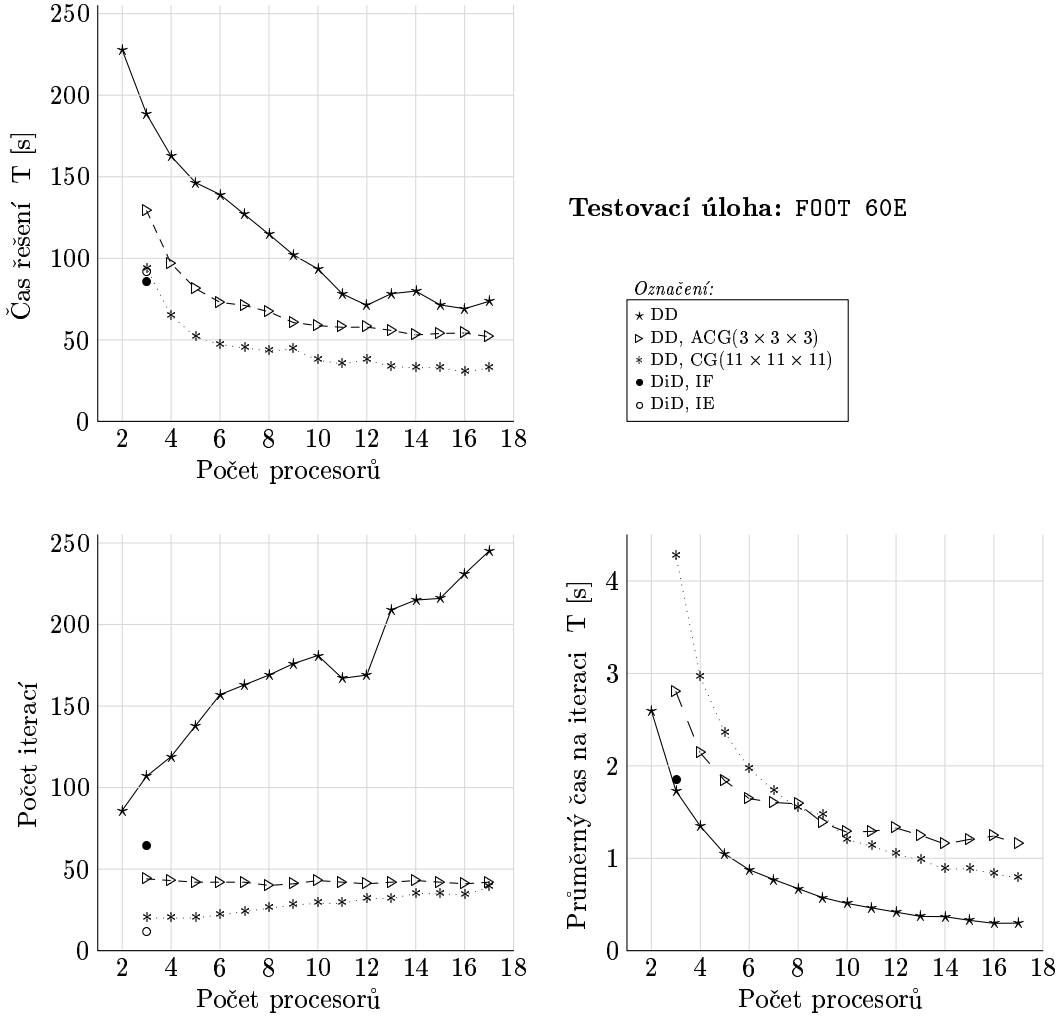


Poznámka: Sekvenční řešič s neúplnou faktorizací: čas řešení 92.7 s, 51 iterací.

Obrázek 40: Testování efektivity paralelních řešičů, SUN-1: FOOT 40 E.

Podle očekávání dosahoval paralelní řešič DD s rostoucím počtem využitých procesorů kratších časů řešení. Celková efektivita řešiče závisela na použitém typu Schwarzova předpodmínění, přičemž metoda sdružených gradientů s dvouúrovňovými předpodmiňovači nalezla řešení rychleji než s jednoúrovňovými. Využití explicitní hrubé sítě znamenalo kvalitativně větší přínos než využití agregované hrubé sítě srovnatelné velikosti. Samotná aplikace hrubé sítě, jak to popisuje te-

orie [18], se příznivě projevila na celkovém počtu iterací potřebných pro nalezení řešení. Zatímco při výpočtu bez hrubé sítě počet iterací rostl s počtem procesorů, při výpočtu s hrubou sítí byl konstantní nebo rostl jen velmi pomalu. Při řešení úlohy FOOT 40 E dosáhl paralelní řešič DD na 7 - 8 procesorech maximální míry využití procesorů, dále již s rostoucím počtem procesorů celkový čas řešení výrazněji neklesal.



Poznámka: Sekvenční řešič s neúplnou faktorizací: čas řešení 397.4 s, 64 iterací.

Obrázek 41: Testování efektivity paralelních řešičů, SUN-1: FOOT 60 E.

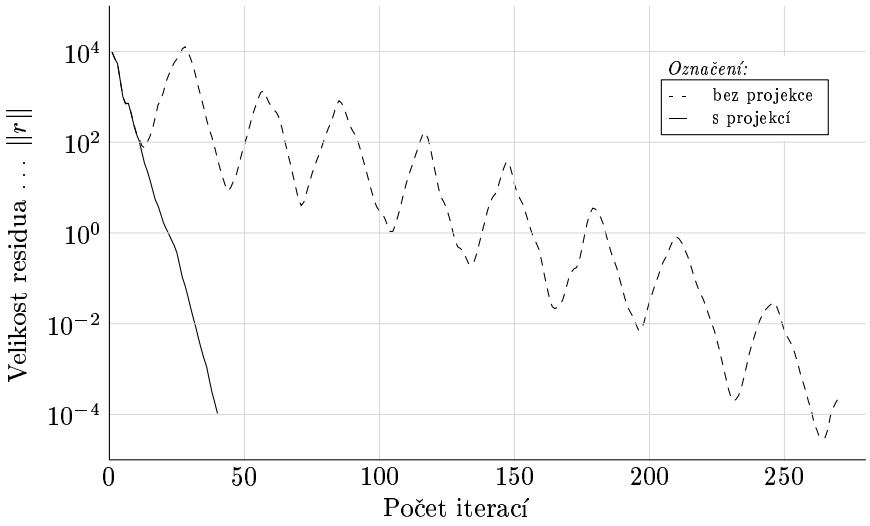
Chování paralelních řešičů se s velikostí řešené úlohy nezměnilo, což dokumentuje obrázek č. 41. Při řešení větší úlohy FOOT 60 E (680 943 rovnic) paralelním řešičem DD se však posunula hranice maximální míry využití procesorů, zejména v případech výpočtů bez hrubé sítě, na celkový počet 10 - 11 procesorů. Pro řešení reálných úloh proto platí, že čím je větší velikost problému, tím je také větší počet procesorů, které lze pro jeho řešení efektivně zaměstnat.

6.4 Projekce pro řešení singulárních úloh

Cílem dalších testů bylo ukázat význam projekce pro řešení lineárních systémů se singulární maticí tuhosti. Nejprve jsme zkoumali vliv projekce na rychlost konvergence metody sdružených gradientů, potom jsme programové implementace algoritmů s projekcí využili při řešení praktické úlohy.

Všechny výpočty byly provedeny na platformě IBM SP. V prvním testu jsme využili paměťově orientovaný řešič `DiD` s předpodmiňovačem založeným na neúplné faktorizaci a testovací úlohu `NEUM 11`, ve druhém testu diskově orientovaný sekvenční řešič se stejným typem předpodmiňovače a paralelní řešič `DiD` s předpodmíněním neúplnou faktorizací nebo vnitřními iteracemi. Druhý test proběhl na testovací úloze `DORO` a výpočty zahrnovaly řešení úplné `NS` nebo `DS` modelovací sekvence. Iterační procesy byly ukončeny dosažením hodnoty relativní přesnosti řešení $\varepsilon = 10^{-3}$ v každém kroku.

Výsledky prvního testu jsou zachyceny na obrázku č. 42. Z grafu závislosti velikosti residua na vykonaném počtu iterací je vidět, že výpočet singulárního systému `NEUM` se po aplikaci projekce stabilizoval a pro redukci velikosti residua vyžadoval několikanásobně menší počet iterací než stejný výpočet bez projekce.



Obrázek 42: Význam projekce pro řešení singulárních úloh, IBM SP: `NEUM 11`.

Praktické výpočty s projekcí shrnuje tabulka č. 6, ve které jsou obsaženy výsledky druhého testu. Přechod od modelovací sekvence `NS` k `DS` se příznivě projevil na celkovém času řešení, jenž v případě sekvenčního řešiče s neúplnou faktorizací poklesl o více než třetinu. Úspora paralelního řešiče `DiD` s neúplnou faktorizací činila více než polovinu času vůči sekvenčnímu řešiči u obou modelovacích sekvencí. Výrazně nejlepšího času dosáhl paralelní řešič `DiD` s vnitřními iteracemi, který čas sekvenčního řešiče překonal téměř 9×! Příčinou je nízký počet vnějších iterací vyžadujících vzájemnou komunikaci dělníků, jenž je na pomalých komunikačních linkách IBM SP časově velmi náročná. Naopak vnitřní iterace nevyžadují

komunikaci dělníků. Vykonávají většinu výpočetní práce, pro níž jsou potřebné pouze diagonální bloky matice tuhosti a proto je zrychlení způsobeno také čtením menšího objemu dat z disku.

DORO – kompletní výpočet				
Modelovací sekvence	Řešič		Čas řešení [hh:mm]	Zrychlení S
	Typ	Vlastnosti		
NS	S	IF	46:16	
DS	S	IF	28:17	
NS	P	DiD, IF	21:20	1.36
DS	P	DiD, IF	13:30	2.10
DS	P	DiD, IE	03:09	**8.98**

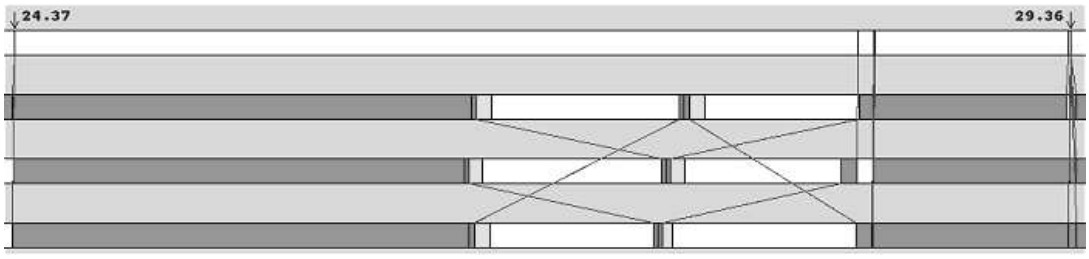
Tabulka 6: Praktické výpočty s projekcí, IBM SP: DORO.

6.5 Rozbor iterací a analýza komunikace

Další test byl zaměřen na rozbor jednotlivých typů paralelních iterací a současně na analýzu komunikace paralelních úloh, která může být klíčová pro výslednou efektivitu paralelního řešiče.

Podmínkou testu bylo omezení pouze na první iteraci[10], kterou každý řešič vykonal při řešení posledního kroku modelovací sekvence DS úlohy DORO na platformě LINUX-4. Při výpočtech řešičem DD byla využita hrubá síť vzniklá agregací uzlů původní sítě, vždy po šesti uzlech v každém směru (AF=6). Běhy řešičů byly zaznamenány do zvláštních souborů prostřednictvím vizualizačního programu XPVM, jenž zároveň umožnil zjistit množství komunikace i časy, po které každý dělník prováděl výpočet, komunikoval nebo čekal.

Na obrázcích s běhy řešičů odpovídá horní záznam vždy běhu řídícího procesu, pod ním následují záznamy běhů jednotlivých dělníků korespondujících s podúlohami. V případě výpočtu s hrubou sítí odpovídá poslední záznam běhu dělníka na hrubé síti.



Obrázek 43: Běh paralelní iterace DiD s neúplnou faktorizací, LINUX-4: DORO.

[10]Každý řešič po svém spuštění nejprve načetl data do paměti, vykonal inicializační fázi, kterou můžeme nazvat *nultou* iterací, a potom pokračoval ve výpočtu jednotlivými iteracemi. První iterace byla předmětem testování.

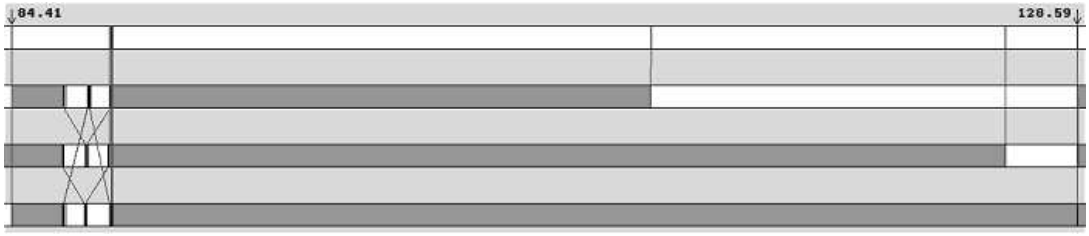
Na obrázcích jsou výpočty znázorněny barvou nejtmavší, systémová režie barvou středně tmavou a čekání barvou nejsvětlejší. Čáry spojující jednotlivé dělníky znázorňují jejich vzájemné komunikace. Šipky s časovými údaji upřesňují počátek a konec první iterace.

Na obrázku č. 43 je zachycen běh paralelní iterace DiD s předpodmíněním neúplnou faktorizací a v tabulce č. 7 rozbor uvedeného typu iterace. Z obrázku je patrné, že řídící proces, sloužící především k vzájemné synchronizaci všech dělníků, je výpočetně nenáročný a jeho funkci by mohl převzít libovolný dělník. Tato výpočetní nenáročnost je pro řídící procesy řešičů DiD a DD charakteristická a proto v dalším textu nebudeme tyto procesy uvažovat.

DORO - poslední krok modelovací sekvence DS				
Výpočet		Komunikace		
Operace	Čas	Operace	Objem dat	Čas
Matice × vektor	2.15 s	Matice × vektor	6× 5164352 b	1.85 s
Předpodmínění	1.01 s	Ostatní	120 b	< 0.02 s
Celkový čas první iterace (3 procesory): **5.01** s				

Tabulka 7: Rozbor paralelní iterace DiD s neúplnou faktorizací, LINUX-4: DORO.

Na celkové době trvání iterace 5.01 s se významně podílela komunikace dělníků během operace matice × vektor, neboť přenos šesti vektorů reálných čísel, každý o velikosti 5164352 bajtů, vyžadoval na Fast Ethernet rozhraní 1.85 s (37 % celkového času). Proto na paralelních počítačích vybavených výkonově srovnatelnými procesory a extrémně rychlými komunikačními linkami, realizovanými například sdílenou pamětí, můžeme čekat zrychlení řešiče až o 35 %. Předpodmínění neúplnou faktorizací nepotřebovalo žádnou komunikaci dělníků a trvalo stejnou dobu 1.01 s (20 % celkového času) u každého dělníka.



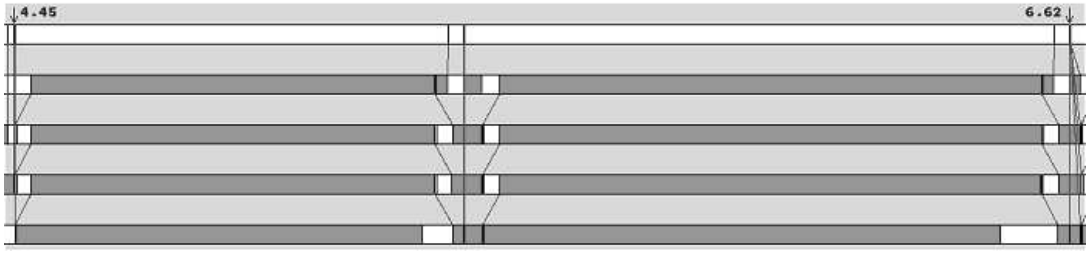
Obrázek 44: Běh paralelní iterace DiD s vnitřními iteracemi, LINUX-4: DORO.

Obrázek č. 44 znázorňuje běh a tabulka č. 8 rozbor paralelní iterace DiD s předpodmíněním vnitřními iteracemi. Je vidět, že předpodmínění je příčinou změny chování tohoto typu iterace, neboť většinu výpočetní práce vykonávaly vnitřní iterace a z celkového času 44.18 s potřebovaly 40.07 s (91 %). Proto u takových výpočtu značně klesá celkový počet vnějších iterací a tím se redukuje i režie spojená s časově náročnou komunikací během operace matice × vektor.

DORO - poslední krok modelovací sekvence DS				
Výpočet		Komunikace		
Operace	Čas	Operace	Objem dat	Čas
Matice × vektor	2.18 s	Matice × vektor	6× 5164352 b	1.92 s
Předpodmínění	40.07 s	Ostatní	120 b	< 0.02 s
Celkový čas první iterace (3 procesory): **44.18** s				

Tabulka 8: Rozbor paralelní iterace DiD s vnitřními iteracemi, LINUX-4: DORO.

Z obrázku č. 44 je patrné, že vnitřní iterace u jednotlivých dělníků nekončí ve stejnou dobu. Předpodmínění je ukončeno až v okamžiku dokončení vnitřních iterací posledním dělníkem, na kterého ostatní dělníci musejí čekat. Toto neefektivní čekání je možné eliminovat tím, že všichni dělníci budou vykonávat vnitřní iterace do okamžiku, než poslední z nich dosáhne požadované relativní přesnosti řešení ε^*. Potom lze očekávat určité zlepšení předpodmiňovače.



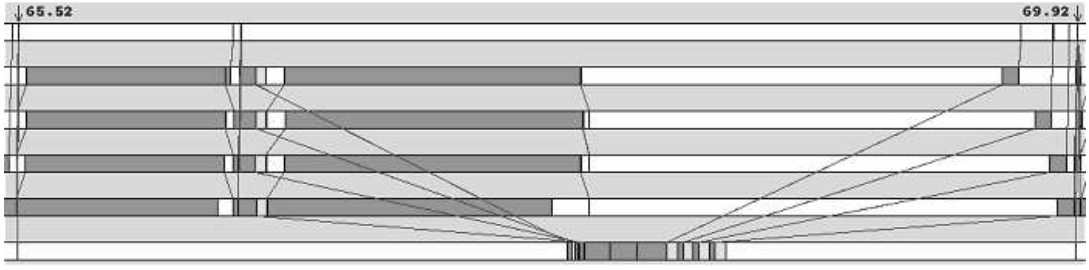
Obrázek 45: Běh paralelní iterace DD, LINUX-4: DORO.

DORO - poslední krok modelovací sekvence DS				
Výpočet		Komunikace		
Operace	Čas	Operace	Objem dat	Čas
Matice × vektor	0.83 s	Matice × vektor	6× 205368 b	0.07 s
Předpodmínění	1.11 s	Předpodmínění	6× 203856 b	0.07 s
Ostatní	0.04 s	Ostatní	160 b	< 0.02 s
Celkový čas první iterace (4 procesory): **2.16** s				

Tabulka 9: Rozbor paralelní iterace DD, LINUX-4: DORO.

Paralelní iterace DD, jejíž běh znázorňuje obrázek č. 45 a rozbor tabulka č. 9, pracovala s jednoúrovňovým aditivním Schwarzovým předpodmiňovačem, kdy jednotliví dělníci řešili podúlohy neúplnou faktorizací. Iterace potřebovala na čtyřech procesorech celkově 2.16 s, z toho na výpočet asi 1.98 s (92 % celkového času) a na komunikaci přibližně 0.16 s (7 % celkového času).

Z obrázku č. 45 je vidět vyváženost rozdělení celkového objemu výpočtů mezi dělníky. Na rozdíl od iterací DiD je celkový objem komunikací podstatně menší, což umožnilo jejich rychlé provádění a redukci časových prodlev při vzájemné interakci dělníků. Celková efektivita tohoto typu iterací je však nižší, což dokumentují výsledky detailního srovnání výkonů řešičů z oddílu 6.3.



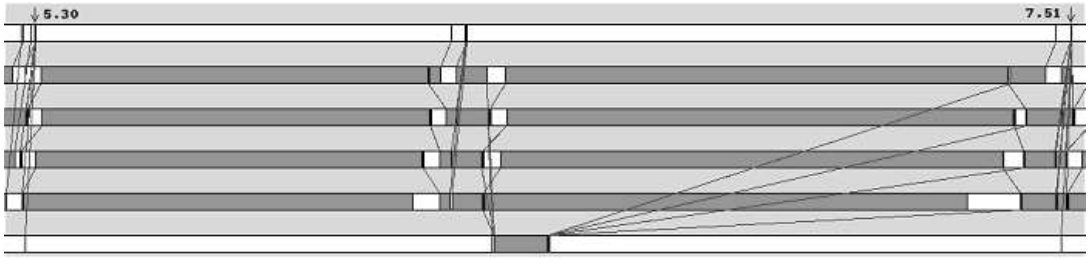
Obrázek 46: Běh původní paralelní iterace DD s hrubou sítí, LINUX-4: DORO.

DORO - poslední krok modelovací sekvence DS					
Výpočet		Komunikace			Čekání
Operace	Čas	Operace	Objem dat	Čas	Čas
Matice × vektor	0.89 s	Matice × vektor	6× 205368 b	0.07 s	*1.71* s
Předpodmínění	1.27 s	Předpodmínění	6× 203856 b	0.07 s	
Ostatní	0.16 s	Ostatní	164 b	< 0.02 s	
Hrubá síť	0.35 s	S podoblastmi	*8× 3873264* b	*2.97* s	1.07 s
Celkový čas první iterace (5 procesorů): **4.40** s					

Tabulka 10: Rozbor původní paralelní iterace DD s hrubou sítí, LINUX-4: DORO.

Vyšší efektivity dosahují paralelní iterace DD s hrubou sítí, které využívají dvouúrovňový aditivní Schwarzův předpodmiňovač. Obrázek č. 46 znázorňuje běh a tabulka č. 10 rozbor původní iterace tohoto typu, jenž neměla minimalizové množství přenášených dat během vzájemné komunikace dělníka na hrubé síti s ostatními dělníky.

Při iteraci, která vyžadovala 4.40 s, dělník na hrubé síti sám výpočetně zajišťoval přechod z jemné na hrubou síť a obráceně, což při čtyřech podoblastech vyžadovalo přenos osmi vektorů reálných čísel, každý o velikosti 3 873 264 bajtů. Tato náročná komunikace trvala celkem 2.97 s a byla příčinou toho, že dělníci pracující na podoblastech museli čekat 1.71 s (39 % celkového času) na dokončení předpodmínění. Čím nižší je rychlost komunikačních linek paralelního počítače, tím se čekání dělníků prodlužuje a naopak. Proto například u symetrických multiprocesorů, jejichž extrémně rychlé komunikační linky realizuje sdílená paměť. se zmíněné čekání dělníků prakticky neprojeví.



Obrázek 47: Běh nové paralelní iterace DD s hrubou sítí, LINUX-4: DORO.

Uvedené čekání lze eliminovat tím, že přechod mezi jemnou a hrubou sítí a zpět budou místo dělníka na hrubé síti realizovat dělníci pracující s podoblastmi. V takovém případě se redukuje objem nezbytné komunikace mezi dělníkem na hrubé síti a ostatními dělníky 90×, celkově na osm vektorů reálných čísel, každý o velikosti 42 840 bajtů. Obrázek č. 47 ukazuje běh a tabulka č. 11 rozbor nové iterace DD s hrubou sítí, která má optimalizovaný objem komunikací. Tato relativně jednoduchá úprava iterace snížila její celkový čas ze 4.40 s na 2.20 s (o 50 %) a zvýšila její efektivitu na paralelních počítačích s pomalejšími komunikacemi, například na klastrech.

DORO - poslední krok modelovací sekvence DS					
Výpočet		Komunikace			Čekání
Operace	Čas	Operace	Objem dat	Čas	Čas
Matice × vektor	0.86 s	Matice × vektor	6× 205368 b	0.08 s	
Předpodmínění	1.13 s	Předpodmínění	6× 203856 b	0.08 s	
Ostatní	0.26 s	Ostatní	164 b	0.03 s	
Hrubá síť	0.11 s	S podoblastmi	*8× 42840* b	0.02 s	2.08 s
Celkový čas první iterace (5 procesorů): **2.20** s					

Tabulka 11: Rozbor nové paralelní iterace DD s hrubou sítí, LINUX-4: DORO.

Ze záznamu a rozboru běhu dělníka na hrubé je však vidět, že tento proces je málo efektivní, neboť po dobu 2.08 s (95 % celkového času) čeká na komunikaci nebo výpočet. Přesto se při praktických výpočtech hrubá síť vyplatí, neboť zvyšuje výkon tohoto typu iterace přibližně 2×, což dokumentují výsledky z oddílu 6.3. Proto předmětem dalšího možného zlepšení iterace bude nalezení vhodné techniky, jak zaměstnat dělníka na hrubé síti a využít jeho výpočetní potenciál k tomu, aby se výkon iterace dále zvýšil.

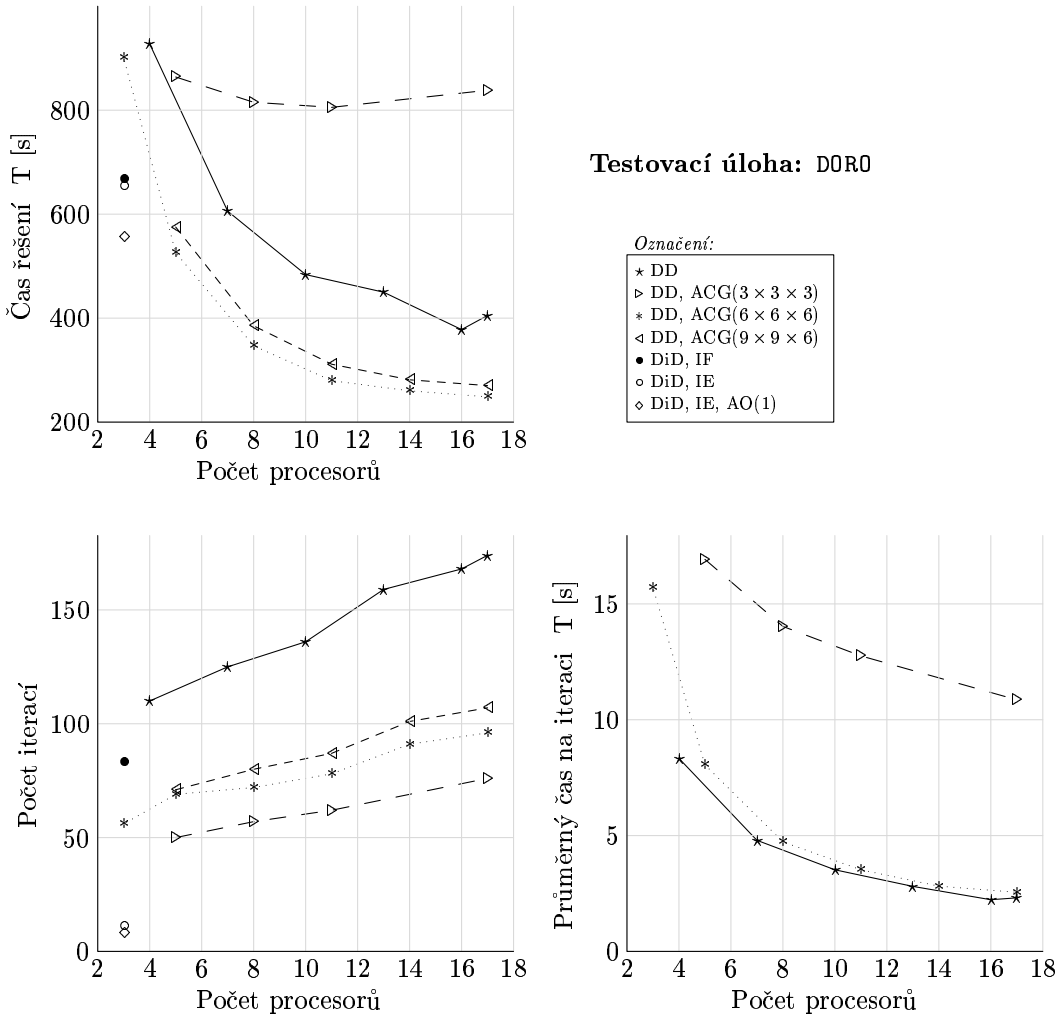
6.6 Řešení náročné úlohy z praxe

Zkušenosti s paralelními výpočty jsme zhodnotili v poslední skupině testů, které jsme zaměřili na řešení praktické úlohy. Současně jsme zjišťovali rozdíly mezi výpočty provedenými na drahém a moderním symetrickém multiprocesoru a na mnohem levnějších klastrech, jejichž základem jsou běžně dostupné personální počítače.

Pro paralelní počítaní jsme zvolili úlohu DORO, u níž jsme řešili pouze poslední krok modelovací sekvence DS. Všechny výpočty byly provedeny paměťově orientovanými řešiči nejprve na platformě SUN-1, později byly některé z nich zopakovány na platformách LINUX-3 a LINUX-4. Při testování jsme využili paralelní řešič DiD s neúplnou faktorizací (IF) nebo s vnitřními iteracemi (IE), doplněný o částečnou dodatečnou ortogonalizaci (AO) k jednomu poslednímu směru postupu, a paralelní řešič DD bez hrubé sítě nebo s agregovanou hrubou sítí (ACG). Hustota hrubé sítě se odvíjela od zvolených agregačních faktorů, které byly po řadě 3×3×3 pro

nejhustší, 6×6×6 pro středně hustou a 9×9×6 pro nejméně hustou agregovanou hrubou síť.

Na obrázku č. 48 jsou výsledky prvního testu, které potvrdily předpokládané a na modelových úlohách ověřené chování paralelních programů. Při výpočtech paralelním řešičem `DiD` s neúplnou faktorizací a s vnitřními iteracemi byl rozdíl v celkových dobách řešení minimální. Pokud však byla algoritmická varianta s vnitřními iteracemi doplněna o výpočetně nenáročnou dodatečnou ortogonalizaci k jednomu poslednímu směru postupu, zkrátil se celkový čas řešení o 15 %, neboť klesl celkový počet iterací potřebných pro nalezení řešení (z 11 na 8).



Poznámka: Sekvenční řešič s neúplnou faktorizací: čas řešení 2959.7 s, 83 iterací.

Obrázek 48: Řešení praktické úlohy, SUN-1: DORO.

Při výpočtech paralelním řešičem DD jsme zkoušeli využít agregovanou hrubou síť různé hustoty. Program pracující se středně hustou hrubou sítí dosáhl na daném počtu procesorů nejkratšího času řešení. Přitom vyžadoval větší počet iterací, než když pracoval s nejhustší hrubou sítí, se kterou dosáhl dokonce nejdelší doby řešení, neboť byl pomalejší než výpočet bez hrubé sítě! Z výsledků je proto patrná důležitost vyvážení objemů výpočetní práce na hrubé síti a na jednotlivých podoblastech tak, aby efektivita řešiče byla maximální. Potom je možné pro řešení úlohy DORO paralelním programem DD efektivně využít 16 - 17 procesorů.

DORO – poslední krok modelovací sekvence DS					
Řešič		It	SUN-1	LINUX-3	LINUX-4
Typ	Vlastnosti		T [s]	T [s]	T [s]
P	DiD, IF	83	668	2653	403
P	DiD, IE	11	653	918	304
P	DiD, IE, AO(1)	8	557		259

Tabulka 12: Paralelní výpočty řešičem DiD, SUN-1, LINUX-3 a LINUX-4: DORO.

Paralelní řešič DiD, primárně určený pro provoz na malých paralelních počítačích (3 - 4 procesory), umožnil rovněž srovnání nákladného symetrického multiprocesoru s levnějšími řešeními na bázi klastrů sestavených z běžných kancelářských počítačů. V tabulce č. 12 jsou shrnuty výsledky testu. Řešič s předpodmíněním neúplnou faktorizací běžel na platformě SUN-1 téměř 4× rychleji než na platformě LINUX-3, což zapříčinil rozdíl v rychlostech a přenosových kapacitách komunikačních linek obou paralelních počítačů. Při využití předpodmínění vnitřními iteracemi se však počet vnějších iterací a tím i objem všech komunikací výrazně zmenšil, což se projevilo ve značném poklesu rozdílu časů řešení na obou platformách (místo 4× pouze 1.4×). Výhody paralelního počítání na klastrech dále umocňují výsledné časy řešiče dosažené na platformě LINUX-4, které jsou přibližně o $40-55\,\%$ kratší než časy řešiče z platformy SUN-1. Přitom pomalejší komunikační linky klastru zvýraznily rozdíl mezi předpodmíněními neúplnou faktorizací a vnitřními iteracemi a zároveň se potvrdila účinnost dodatečné ortogonalizace.

DORO – poslední krok modelovací sekvence DS					
Řešič		Počet	It	SUN-1	LINUX-4
Typ	Vlastnosti	procesorů		T [s]	T [s]
P	DD, ACG(6×6×6)	3	56	900	242
P	DD, ACG(6×6×6)	5	64	526	145
P	DD, ACG(6×6×6)	8	72	347	111

Tabulka 13: Paralelní výpočty řešičem DD, SUN-1 a LINUX-4: DORO.

Rostoucí schopnosti klastrů personálních počítačů výpočetně konkurovat paralelním počítačům s architekturou speciálně navrženou pro vysoce výkonné počítání potvrzují testy řešiče DD, kdy jsme výpočty s nejlepšími výsledky dosažené na platformě SUN-1 zopakovali na platformě LINUX-4. Výsledky testu, při kterých řešič na klastru ve všech případech rychlostně více než 3× překonal řešič na symetrickém multiprocesoru, shrnuje tabulka č. 13.

Po uvážení nákladů na pořízení, provoz a údržbu uvedených paralelních počítačů se proto právem dostávají do popředí zájmů uživatelů levná řešení. Jejich typickým zástupcem může být právě klastr osobních počítačů běžících pod operačním systémem typu UNIX (Linux, FreeBSD nebo jiný). Vhodný návrh a následná implementace paralelního algoritmu potom dovolují očekávat výborné výsledky a úsporu nemalých finančních prostředků.

7 Závěr

Předložená práce se týká návrhů a implementací efektivních paralelních řešičů rozsáhlých soustav lineárních rovnic, které vznikají diskretizací 3D okrajových úloh pružnosti metodou konečných prvků. Základem řešičů se stala metoda sdružených gradientů s předpodmíněním a metody rozkladu prostoru, umožňující dva různé typy dekompozice datových struktur uvažovaného problému: separace složek posunutí `DiD` a rozklad oblasti s překrytím `DD`.

Po nezbytném seznámení s problematikou řešení úloh pružnosti, prostředky paralelizace i potřebnými numerickými metodami jsem dlouhodobým vývojem vytvořil několik variant řešičů. Paralelní verze řešičů jsou univerzální, mohou pracovat buď v prostředí PVM nebo MPI, které mají v dnešní době své implementace na většině paralelních počítačů. Zdrojové texty řešičů i všech souvisejících programů z konečněprvkového systému `PortaGEM` jsou přenositelné, proto mohou spustitelné tvary programů pracovat na různých platformách paralelních výpočetních systémů. Důkazem přenositelnosti programů jsou experimenty[11] provedené na počítačích instalovaných na pracovištích: EPCC Edinburgh, KTH Stockholm, CLPP BAS Sofia, SARA Amsterdam, VŠB - TU Ostrava a ÚGN AV ČR Ostrava. Programy jsem prakticky zkoušel na testovacích úlohách z geomechaniky i při výpočtu reálné úlohy velkého rozsahu. Řešiče při všech výpočtech potvrdily svoji dobrou výslednou efektivitu.

Kromě řešení úloh (lineární) pružnosti lze předloženou práci rozšířit a doplnit o řešení řady dalších problémů, například úloh nelineární pružnosti, termopružnosti nebo časových úloh. Výsledky a poznatky je možné rovněž aplikovat při vývoji dalších řešičů, které využívají jiné numerické metody, například předpodmínění Schurovým doplňkem, FETI metody, multigridní metody a další, nebo které jsou určeny pro speciální třídu paralelních počítačů, například pro heterogenní počítačové klastry.

Conclusion

The thesis are devoted to designs and implementations of the efficient parallel solvers for large-scale linear systems arising from the finite element discretization of elasticity problems in 3D. The presented solvers are based on the preconditioned conjugate gradient method and on the space decomposition methods. Especially two possible techniques enabling to split data structures of the solved problem are used: the displacement decomposition and the domain decomposition.

The parallel solvers are universal in the sense that exploit the Parallel Virtual Machine or the Message Passing Interface, which have their implementations on

[11] Práce zahrnuje pouze vybrané experimenty.

the most of nowaday parallel computers. The source codes of the solvers and also of all related programs belonging to the finite element system PortaGEM are portable and therefore their executable codes can run on various platforms of parallel computer systems. The portability was proved by testing of the solvers on parallel computers installed at EPCC Edinburgh, KTH Stockholm, CLPP BAS Sofia, SARA Amsterdam, TU Ostrava and IGAS Ostrava. The programs were tested on benchmark problems in geomechanics and at a solution of real-life large-scale problem. The solvers showed the good resulting efficiency at all performed computations.

Except the solution of (linear) elasticity problems, the work joined to the thesis can be extended to the solution of many other problems such as nonlinear elasticity, thermoelasticity or time-dependent problems, for instance. Achieved results and experience can be also applied to a development of further solvers, which exploit other numerical methods for example preconditioning by the Schur complement, FETI methods, multigrids, etc. or which are intended for a special class of parallel computers such as heterogeneous computer clusters and others.

Literatura

[1] M. R. Hestenes, E. Stiefel: *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards 49/6, 1952.

[2] J. K. Reid: *On the method of conjugate gradients for the solution of large sparse systems of linear equations.* Large Sparse Sets of Linear Equation, Academic Press, London, 1971.

[3] M. J. Flynn: *Some computer organizations and their effectiveness.* IEEE Transactions on Computers C-21, 1972.

[4] J. A. Meijerink, H. A. van der Vorst: *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.* Mathematics of Computations 31, 1977.

[5] O. Axelsson, I. Gustafsson: *Iterative methods for the solution of the Navier equations of elasticity.* Computer Methods in Applied Mechanics and Engineering 15, 1978.

[6] J. Nečas, I. Hlaváček: *Úvod do matematické teorie pružných a pružně plastických těles.* SNTL - Nakladatelství technické literatury, Praha, 1983.

[7] O. Axelsson, V. A. Barker: *Finite Element Solution of Boundary Value Problems. Theory and Computations.* Computer Science and Applied Mathematics, Academic Press, 1984.

[8] R. Glowinski, G. H. Golub, G. A. Meurant, J. Périaux: *The First International Symposium on Domain Decomposition Methods for Partial Differential Equations.* SIAM, Philadelphia, 1988.

[9] T. F. Chan, R. Glowinski, J. Périaux, O. B. Widlund: *Domain Decomposition Methods.* SIAM, Los Angeles, 1989.

[10] J. Stoer, R. Burlisch: *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1993.

[11] O. Axelsson: *Iterative Solution Methods.* Cambridge University Press, 1994.

[12] R. Blaheta: *Displacement decomposition – incomplete factorization preconditioning techniques for linear elasticity problems.* Numerical Linear Algebra with Application 1, 1994.

[13] A. Geist et al.: *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, Cambridge, 1994.

[14] R. Blaheta, R. Kohut: *PCG–1 iterative solver.* Zpráva 9501, Ústav geoniky AV ČR, Ostrava, 1995.

[15] I. T. Foster: *Designing and Building Parallel Programs.* Concepts and Tools for Parallel Software Engineering, Addison-Wesley Publishing, 1995.

[16] *MPI: A Message-Passing Interface Standard.* Message Passing Interface Forum, University of Tennessee, Knoxville, 1995.

[17] J. A. Kohl, A. Geist: *XPVM 1.0 User's guide.* Oak Ridge National Laboratory, Tennessee, 1996.

[18] B. Smith, P. Bjørstad, W. Gropp: *Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.

[19] J. Starý: *Využití PVM pro paralelní implementaci metody sdružených gradientů s předpodmíněním.* Diplomová práce, VŠB – Technická univerzita, Ostrava, 1996.

[20] R. Blaheta, O. Jakl, J. Starý: *A parallel CG solver for FE analysis of 3D problems in geomechanics.* Geomechanics '96, Rožnov pod Radhoštěm, 1996.

[21] R. Blaheta, O. Jakl, R. Kohut, A. Kolcun: *An application of large scale mathematical moddeling in geomechanics.* Sborník konference Moderní matematické metody v inženýrství, Ostrava, 1997.

[22] K. Ježek, P. Matějovic, S. Racek: *Paralelní architektury a programy.* Vydavatelství Západočeské univerzity, Plzeň, 1997.

[23] J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Vorst: *Numerical Linear Algebra for High-Performance Computers.* SIAM, Philadelphia, 1998.

[24] O. Axelsson, I. Kaporin: *Error norm estimation and stopping criteria in preconditioned conjugate gradient iterations.* Report 6, University of Nijmegen, 2000.

[25] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith: *PETSc Users Manual.* Revision 2.1.1. Argonne National Laboratory, 2001.

[26] D. Braess: *Finite Elements. Theory, fast solvers, and applications in solid mechanics.* Cambridge University Press, second edition, 2001.

[27] MPICH Home Page: *MPICH - A Portable Implementation of MPI,* `http://www.mcs.anl.gov/mpi/mpich/` (19.2.2002)

[28] LAM Home Page: *LAM/MPI Parallel Computing*, `http://www.lam-mpi.org/` (19.2.2002).

[29] R. Blaheta: *GPCG – generalized preconditioned CG method and its use with nonlinear and nonsymmetric displacement decomposition preconditioners.* Numerical Linear Algebra with Applications 9, 2002.

[30] R. Blaheta: *Nové možnosti matematického modelování v geomechanice při řešení 3D úloh velkého rozsahu.* Uhlí, rudy, geologický průzkum 6, 2002.

[31] *HPL – A Portable Implementation of the High-Performance LINPACK Benchmark for Distributed-Memory Computers*, `http://www.netlib.org/benchmark/hpl` (30.4.2003).

[32] *The NAS Parallel Benchmarks*, `http://www.nas.nasa.gov/Software/NPB` (30.4.2003).

Autorské práce

- J. Starý: *Využití PVM pro paralelní implementaci metody sdružených gradientů s předpodmíněním.* Diplomová práce, VŠB – Technická univerzita, Ostrava, 1996.
- R. Blaheta, O. Jakl, J. Starý: *Paralelizace metody sdružených gradientů.* Programy a algoritmy numerické matematiky, Janov nad Nisou, 1996.
- R. Blaheta, O. Jakl, J. Starý: *A parallel CG solver for FE analysis of 3D problems in geomechanics.* Geomechanics '96, Rožnov pod Radhoštěm, 1996.
- R. Blaheta, O. Jakl, J. Starý: *Zkušenosti s paralelními výpočty na IBM SP.* Moderní matematické metody v inženýrství, Nová Ves u Frýdlantu nad Ostravicí, 1997.
- R. Blaheta, O. Jakl, J. Starý: *Parallel implementation of the displacement decomposition preconditioning for elasticity.* Iterative methods and parallel computing, Milovy, 1997.
- R. Blaheta, O. Jakl, J. Starý: *PVM-implementation of the PCG method with displacement decomposition.* The fourth European PVM-MPI User's Group Meeting, Krakow, 1997.
- O. Jakl, J. Starý: *Paralelní výpočty v prostředí MPI.* Moderní matematické metody v inženýrství, Nová Ves u Frýdlantu nad Ostravicí, 1998.
- R. Blaheta, O. Jakl, J. Starý: *An application of high performance computing in geomechanics.* Moderní matematické metody v inženýrství, Nová Ves u Frýdlantu nad Ostravicí, 1998.
- J. Starý: *Paralelní implementace řešičů pro úlohy pružnosti na základě separace složek posunutí: nástroje, algoritmy a efektivita paralelizace.* Rigorózní práce, VŠB – Technická univerzita, Ostrava, 1998.
- R. Blaheta, O. Jakl, J. Starý: *Large-scale FE modelling in geomechanics: a case study in parallelization.* The sixth European PVM-MPI User's Group Meeting, Sabadell, Barcelona, 1999.
- R. Blaheta, O. Jakl, R. Kohut, J. Starý: *Iterative displacement decomposition solvers for HPC in geomechanics.* The second workshop Large-Scale Scientific Computations, Sozopol, 1999.
- R. Blaheta, O. Jakl, R. Kohut, A. Kolcun, J. Starý: *An analysis of the stress fields induced by mining with application of parallel high performance computing.* Numerical Models in Geomechanics, Graz, 1999.

- R. Blaheta, J. Starý: *Paralelní implementace metody sdružených gradientů s předpodmíněním pomocí Schwarzových metod.* Technická zpráva, Ústav goniky AV ČR, Ostrava, 2000.
- J. Starý: *Using the Schwarz methods for parallel solution of elasticity problems.* Paralelní algoritmy a numerické metody 10, Matematický ústav AV ČR, Libverda, 2000.
- R. Blaheta, O. Jakl, J. Starý: *Library of parallel PCG solvers for problems of geomechanics.* Algoritmy, Podbánské, 2000.
- R. Blaheta, O. Jakl, J. Starý: *Library of parallel PCG solvers*, (version 1.0 draft). Technická zpráva, Ústav geoniky AV ČR, Ostrava, 2001.
- R. Rlaheta, O. Jakl, R. Kohut, J. Starý: *Large scale FE analysis of stresses due to excavation and heating.* Geonics 2001, Ostrava, 2001.
- R. Rlaheta, P. Byczanski, O. Jakl, J. Starý: *Space decomposition preconditioners and their application in geomechanics.* The second IMACS Conference on mathematical modelling and computational methods in mechanics, physics, biomechanics and geodynamics. Plzeň, 2001.
- J. Starý: *Solvers for large linear systems arising from the FE discretization of elasticity problems.* Introduction to High-Performance Computing, Summer school. Stockholm, 2001.
- R. Blaheta, O. Jakl, J. Starý: *Displacement decomposition and parallelization of the PCG method for elasticity problems.* International Conference on Parallel Processing, Valencia, 2001.
- R. Blaheta, O. Jakl, J. Starý: *Parallel displacement decomposition solvers for elasticity problems.* Parallel Processing and Applied Mathematics, Naleczow, 2001.
- R. Blaheta, O. Jakl, K. Krečmer, J. Starý: *Large-scale modelling in geomechanics with parallel computing on clusters of PC's.* NUMGE'02 The fifth European Conference Numerical Methods in Geotechnical Engineering, Paris, 2002.
- R. Blaheta, P. Byczanski, O. Jakl, R. Kohut, A. Kolcun, K. Krečmer, J. Starý: *Application of large-scale FEM computations for design of new extracting technologies in mining.* Technická zpráva, Ústav geoniky AV ČR, Ostrava, 2002.
- R. Blaheta, O. Jakl, J. Starý: *Parallel high-performance computing in geomechanics with inner/outer iterative procedures.* International Conference on Computational Science, Amsterdam, 2002.

- A. Hájek, P. Konečný, R. Šňupárek a další: *Chování horského masívu při dobývání velkých rudních těles na uranovém ložisku Rožná.* Technická zpráva, Ústav geoniky AV ČR, Ostrava, 2002.

- R. Blaheta, P. Byczanski, O. Jakl, J. Starý: *Space decomposition preconditioners and their application in geomechanics.* Mathematics and Computers in Simulation 61, 2003.

- R. Blaheta, P. Byczanski, J. Starý: *Large scale modelling in geomechanics: elasticity, thermo-elasticity and iterative solvers.* Mathematical and Computer Modelling in Science and Engineering, Praha, 2003.

- K. Krečmer, J. Starý: *Use of PETSc for high-performance computing in geomechanics.* Technická zpráva, Ústav geoniky AV ČR, Ostrava, 2003.

- R. Blaheta, O. Jakl, J. Starý: *Linear system solvers based on space decompositions and parallel computations.* Inženýrská mechanika, bude vydáno.