VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Využití PVM pro paralelní implementaci metody sdružených gradientů s předpodmíněním.

Diplomová práce

1996 Jiří Starý

Místopřísežně prohlašuji, že jsem celou diplomovou práci vypracoval samostatně a pouze s použitím literatury, kterou jsem uvedl v seznamu.

Datum odevzdání: Podpis:

# 1 Úvod

Pro náročnější technické výpočty je nutné použít výkonnější výpočetní prostředek než běžný osobní počítač nebo pracovní stanici. Dříve k takovým účelům bylo možné použít pouze superpočítač, v dnešní době už existuje více možností.

Jednou z nich je použití paralelních počítačů, které v posledních letech zaznamenaly značný rozvoj. Paralelní počítače se skládají z několika stejných komponent (obvykle myslíme procesory, tuto variantu budeme uvažovat), které mohou pracovat současně. Tyto počítače nedosahují takových špičkových výkonů jako superpočítače, mají však daleko příznivější poměr cena/výkon, což je hlavní příčinou jejich rozvoje.

Výkon paralelního počítače může být značně variabilní podle toho, o jaký typ paralelity se jedná. Základní rozdělení paralelních počítačů můžeme provést podle vztahu procesorů k operační paměti: pokud je operační paměť společná všem procesorům, hovoříme o systémech se sdílenou pamětí, pokud má každý procesor svou vlastní paměť, hovoříme o systémech s distribuovanou pamětí.

Systémy se sdílenou pamětí jsou konstrukčně náročnější, neboť musí být vhodně vyřešeny styk mezi jednotlivými procesory a styk mezi procesory a pamětí. Tato konstrukční náročnost ovlivňuje kromě ceny systému také maximální počet procesorů (méně než 20).

V systémech s distribuovanou pamětí má každý procesor svoji paměť, proto jsou konstrukčně méně náročné, což snižuje jejich cenu a rozšiřuje počet jejich uživatelů. Tato architektura je výhodná z hlediska maximálního počtu procesorů, který není principiálně omezen. Naopak nevýhodné jsou pomalá komunikace mezi procesory a zejména přístup procesoru k datům, která nemá ve své paměti. Pokud je takových přístupů při řešení problému větší počet, rychlost řešení velmi klesá.

Velká pozornost se v poslední době věnuje skupině spřažených pracovních stanic, na které lze pohlížet jako na paralelní počítač s distribuovanou pamětí. Systémům tohoto typu se říká klastry (clustery), farmy, . . . . Systém může být realizován pomocí softwaru, který je instalován na několik pracovních stanic, které jsou připojeny k lokální síti. Takovým softwarem může být např. balík PVM (Parallel Virtual Machine), který vytváří prostředí pro běh paralelních

programů na klastru pracovních stanic. Při vhodné paralelní implementaci řešení problému tak může klastr provádět náročné technické výpočty.

Takovým výpočtem může být řešení soustavy lineárních rovnic, které vznikají diskretizací úloh 3D pružnosti a plasticity metodou konečných prvků. Tyto soustavy jsou velmi rozsáhlé, běžně dosahují stovek tisíců rovnic, a to je také důvod, který brání nalezení řešení soustavy použitím přímých metod. Proto je problém vyřešení soustavy lineárních rovnic jedním z nejdůležitějších úkolů numerické matematiky.

Řešením problému se jeví použití iteračních metod, které mají z hlediska paměťové náročnosti mnohem lepší vlastnosti než přímé metody. Prakticky se používá metoda sdružených gradientů, u které se rychlost konvergence metody zvyšuje dalšími technikami, mezi které patří předpodmínění nebo vhodná počáteční aproximace.

V diplomové práci se budeme zabývat efektivní implementací metody sdružených gradientů na klastru pracovních stanic. Součástí práce je vytvoření sekvenčního a paralelního programu pro řešení soustavy lineárních rovnic metodou sdružených gradientů s využitím separace složek posunutí datových struktur. Programy zároveň umožní odzkoušet novou variantu počáteční aproximace. V jednotlivých částech práce se budeme zabývat nejprve algoritmizací sekvenční i paralelní verze metody sdružených gradientů, dále prostředky pro realizaci uvedených programů, popisem vytvořených programů, testováním programů na modelových úlohách a posouzením jejich efektivity.

# 2 Sdružené gradienty

Metoda sdružených gradientů je iterační metoda pro řešení soustavy lineárních rovnic

$$Au = f$$

se symetrickou positivně definitní maticí soustavy $A$. Výhodná je pro svou algoritmickou jednoduchost a paměťovou nenáročnost, která je podstatně nižší než u přímých metod.

Rychlost konvergence metody závisí na rozložení vlastních čísel matice soustavy $A$, rychlou konvergenci lze očekávat v případě malého čísla podmíněnosti nebo shluků vlastních čísel daleko od nuly. Ke zlepšení konvergence se využívá technika nazvaná 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 rovné práci pro vykonání jedné iterace vynásobené počtem iterací. Pro metodu sdružených gradientů se v jedné iteraci provádí především součin matice krát vektor, dále je potřeba provést určitý počet operací s vektory (dvakrát skalární součin, třikrát násobení vektoru skalárem a sčítání vektorů). Při použití předpodmínění musíme v každé iteraci navíc řešit soustavu s předpodmiňující maticí.

## 2.1 Algoritmus

Algoritmus metody sdružených gradientů s předpodmíněním je popsán v [1], [2]. Jeho přepis do tvaru, který je vhodný pro implementaci na počítači, je v [4].

Algoritmus pracuje s vektory reálných čísel $u, r, re, v, w$ a výpočet probíhá ve třech fázích:

- $u$ ... hledané řešení
- $r$ ... pravá strana
- $re$ ... residua
- $v$ ... směr postupu (změna vektoru $u$)
- $w$ ... směr postupu (změna vektoru $re$)

**I. Příprava** – nultá iterace (ITER=0), počáteční hodnoty proměnných.

| | | | |
|---|---|---|---|
| $r$ | $\leftarrow$ | $f$ | načtení vektoru pravých stran $r$ z datových souborů, |
| $u$ | $\leftarrow$ | $u_{init}$ | nastavení počáteční aproximace, $u_{init} = 0$, $u_{init} = C^{-1}r$, $u_{init}$ může být načteno z datových souborů nebo může být produktem „částečných řešení“, |
| $w$ | $\leftarrow$ | $Au$ | násobení matice krát vektor, |
| $re$ | $\leftarrow$ | $r - w$ | výpočet výchozích residuí, |
| $v$ | $\leftarrow$ | $C^{-1}re$ | výpočet směru postupu pro hledání řešení, matice $C$ je dána zvoleným typem předpodmínění, |
| $srr0$ | $=$ | $< re, re >$ | |
| $srw0$ | $=$ | $< re, v >$ | |
| $sff$ | $=$ | $< r, r >$ | výpočet skalárních součinů. |

**II. Opakovaný výpočet** – iterace (ITER=1,2,...).

| | | | |
|---|---|---|---|
| $w$ | $\leftarrow$ | $Av$ | násobení matice krát vektor, |
| $s$ | $=$ | $< w, v >$ | |
| $a$ | $=$ | $srw0/s$ | výpočet koeficientu $a$, |
| $u$ | $\leftarrow$ | $u + a * v$ | zpřesnění řešení, |
| $re$ | $\leftarrow$ | $re - a * w$ | nová residua, |
| $w$ | $\leftarrow$ | $C^{-1}re$ | úprava residua, matice $C$ je dána zvoleným typem předpodmínění, |
| $srr$ | $=$ | $< re, re >$ | |
| $srw$ | $=$ | $< re, w >$ | |
| $b$ | $=$ | $srw/srw0$ | výpočet koeficientu $b$, |
| $srw0$ | $=$ | $srw$ | |
| $v$ | $\leftarrow$ | $w + b * v$ | nový směr postupu pro hledání řešení. |

**III. Testování** – následuje po každé iteraci.

| | | | |
|---|---|---|---|
| $tol$ | $=$ | $\sqrt{\frac{srr}{sff}}$ | zjištění dosažené relativní přesnosti výpočtu, |
| $tol$ | $\leq$ | $\varepsilon$ | pokud je dosažená přesnost $\leq$ zadané přesnosti $\varepsilon$, uloží se vektor $u$ do datových souborů a výpočet končí, v opačném případě pokračuje výpočet znovu další iterací. |

## 2.2 Předpodmínění

Předpodmínění slouží ke zlepšení spektrálních vlastností matice tuhosti. Je dáno předpodmiňující maticí $C$. Je možné jej implementovat přechodem k transformované soustavě nebo jak tomu bude v této práci, modifikací residua

$$w = C^{-1}re,$$

tj. řešením soustavy

$$Cw = re$$

v každé iteraci. 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 $A$, potom hovoříme o diagonálním předpodmínění,

- $C = (X - L)X^{-1}(X - L)^{\mathtt{T}}$,

  kde $L$ je dolní trojúhelníková část matice $A$, $X$ je diagonální matice určená podmínkou $Ce = A\ e$, $e = (1, \ldots, 1)^{\mathtt{T}}$, potom hovoříme o předpodmínění DD-IF.

Podrobnější popis možných typů předpodmínění lze nalézt v [3], [5], [6].

## 2.3 Počáteční aproximace

Dalším ze způsobů, kterým lze ovlivnit výpočet řešení soustavy lineárních rovnic metodou sdružených gradientů, je vhodná počáteční aproximace řešení $u_{init}$. Volba počáteční aproximace ovlivňuje především velikost počáteční chyby, může však ovlivnit i rychlost konvergence. Budeme uvažovat následující volby počáteční aproximace $u_{init}$:

- $u_{init} = 0$,

  potom hovoříme o nulové počáteční aproximaci,

- $u_{init} = C^{-1}r$,

  pro metodu sdružených gradientů s předpodmíněním, $C$ je předpodmiňující matice daná typem předpodmínění, $r$ je pravá strana soustavy,

- $u_{init} \leftarrow \tilde{u}_{init}$,

  pro načtení počáteční aproximace z datových souborů, přičemž $\tilde{u}_{init}$ je výsledek dříve provedeného výpočtu ($\tilde{u}_{init}$ může být například produkt jiné numerické metody aplikované na stejnou úlohu).

- $u_{init} \leftarrow u_x, u_y, u_z$,

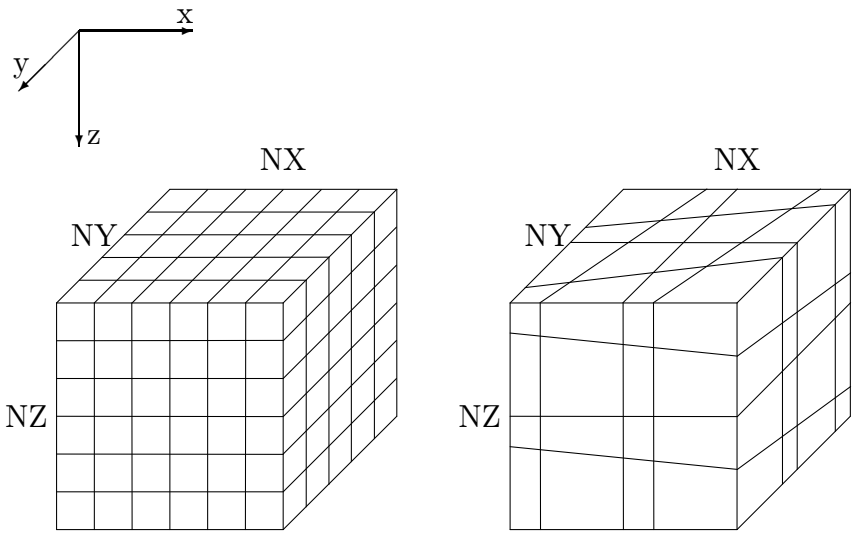
  v této práci bude vyzkoušena další volba počáteční aproximace nazvaná „částečná řešení“, jejíž efektivní implementace vychází z blokového formátu uložení datových struktur úlohy (viz. následující oddíl Uložení matice tuhosti/Blokový formát, obr. 5). Složky počáteční aproximace $u_{init}$ jsou řešením soustav

$$\begin{aligned} A_{11}\ u_x &= r_x, \\ A_{22}\ u_y &= r_y, \\ A_{33}\ u_z &= r_z, \end{aligned}$$

  metodou sdružených gradientů s diagonálním předpodmíněním.

## 2.4 Řešené úlohy

Vlastní implementace řešiče (volba numerické metody, návrh datových struktur, ...) se odvíjí od typu řešených úloh.

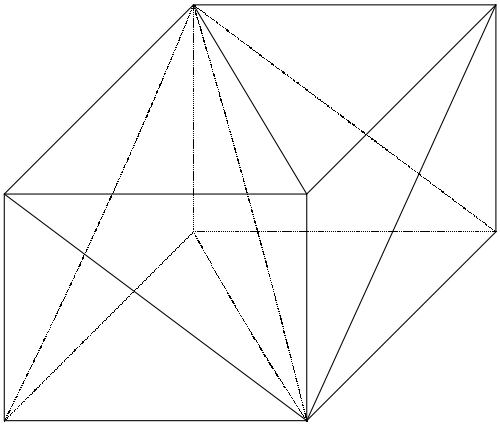



Obr. 1: Pravidelné sítě pro diskretizaci úloh

Soustavy lineárních rovnic, které vznikají diskretizací úloh 3D pružnosti metodou konečných prvků jsou rozsáhlé ($10^3$ – $10^6$ rovnic). Matice soustavy, která se u takto vzniklých soustav nazývá matice tuhosti, je řídká, symetrická, positivně definitní.

Při omezení na pravidelné sítě vznikající při pravidelném dělení oblasti na osmiuzlové „cihly“, které jsou dále děleny na lineární čtyřstěnné prvky lze navíc využít pravidelné uspořádání matice tuhosti k jejímu uložení.

Pro uložení soustavy vzniklé uvedeným způsobem jsou důležité parametry sítě. Patří mezi ně NX, NY, NZ, což jsou počty uzlů sítě po řadě ve směru x, y, z. Celkový počet uzlů sítě je $\mathrm{NN} = \mathrm{NX} * \mathrm{NY} * \mathrm{NZ}$ a celkový počet rovnic pro 3D úlohu je $\mathrm{ND} = 3 * \mathrm{NN}$.



Obr. 2: Rozdělení dílku sítě na lineární konečné prvky

Příklady pravidelných sítí pro diskretizaci 3D úloh jsou znázorněny na obr. 1 a způsob aplikace lineárních konečných prvků ve tvaru čtyřstěnu při rozdělení sítě je zachycen na obr. 2.

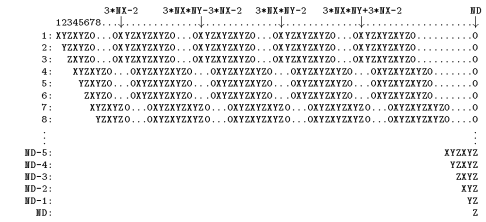
## 2.5 Uložení matice tuhosti

Pro řešení rozsáhlých soustav lineárních rovnic je důležité efektivní uložení datových struktur úlohy, zejména matice tuhosti, z důvodu snížení paměťové náročnosti úlohy. K tomu účelu se využívají charakteristické vlastnosti matice tuhosti.

Matice tuhosti je symetrická, proto stačí ukládat pouze prvky horní trojúhelníkové části matice tuhosti. Při diskretizaci úlohy pravidelnou sítí má matice tuhosti na každém řádku pravidelně umístěné nenulové prvky, přičemž jejich skutečný počet závisí na typu použitých konečných prvků. V případě matice tuhosti uspořádané způsobem, který je popsán v následujícím oddíle, je na každém řádku horní trojúhelníkové části matice maximálně 42 nenulových prvků. Z toho vyplývá, že matice je řídká a do datových souborů se budou ukládat pouze nenulové prvky.

### 2.5.1 Původní formát

Ve [4],[6] je popsáno uložení matice tuhosti, které používá původní sekvenční program PCG–1. Stručný popis vychází ze schématu na obr. 3.



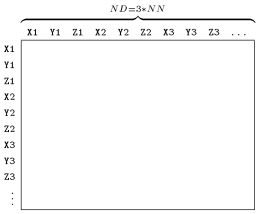

Obr. 3: Původní formát uložení matice tuhosti

Pozice nenulových prvků na řádku matice odpovídají pravidelné šabloně a jsou závislé na parametrech sítě NX, NY a NZ. Celkový počet řádků matice je $\mathrm{ND} = 3 * \mathrm{NX} * \mathrm{NY} * \mathrm{NZ}$, přičemž řádky odpovídající složkám posunutí daného uzlu jsou ukládány za sebou v pořadí x, y a z.

V datovém souboru jsou uloženy pouze 42 nenulové prvky každého řádku horní trojúhelníkové části matice. V rutinách, které pracují s maticí, se inicializuje šablona, což je pole indexů délky 42, na hodnoty odpovídající skutečným pozicím nenulových prvků v prvním řádku matice. Při zpracování řádku matice odpovídajícího posunutí se šablona v závislosti na tomto posunutí aktualizuje. Aby se nemuselo ošetřovat zkrácení posledních řádků matice, jejichž počet je dán poloviční šířkou pásu, ukládají se do datového souboru i tyto řádky v plné délce, tj. 42 prvků. Proto jsou o poloviční šířku pásu zvětšeny i vektory, které se při výpočtech s touto maticí používají. Takto zvýšená paměťová náročnost implementace je poměrně malá, přičemž jsou výrazně zjednodušeny algoritmy pracující s takto uloženou maticí.

### 2.5.2 Blokový formát

Z hlediska paralelizace metody sdružených gradientů je výhodnější separovat složky posunutí uzlů sítě do ucelených bloků podle směru posunutí. Prvky matice tuhosti, jejíž schéma v původním formátu je na obr. 4, jsou přeuspořádány do blokového tvaru, jehož schéma je na obr. 5.

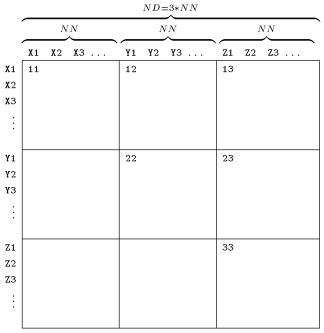



Obr. 4: Schéma původního formátu uložení matice tuhosti

Jednotlivé submatice matice tuhosti jsou uloženy v samostatných datových souborech. Díky symetrii matice tuhosti, tj. $A = A^{\mathtt{T}}$, jsou uloženy pouze prvky z bloků, které jsou v obrázku označeny indexy, protože $\mathrm{A}_{12}^{\mathtt{T}} = \mathrm{A}_{21}$, $\mathrm{A}_{13}^{\mathtt{T}} = \mathrm{A}_{31}$ a $\mathrm{A}_{23}^{\mathtt{T}} = \mathrm{A}_{32}$. Celkový počet řádků každé submatice odpovídá počtu uzlů sítě, tzn. $\mathrm{NN} = \mathrm{NX} * \mathrm{NY} * \mathrm{NZ}$.

Submatice $\mathrm{A}_{11}$, $\mathrm{A}_{22}$ a $\mathrm{A}_{33}$ resp. $\mathrm{A}_{12}$, $\mathrm{A}_{13}$ a $\mathrm{A}_{23}$ mají shodné umístění prvků na řádku, které odpovídá pravidelné šabloně a je závislé na parametrech sítě NX, NY a NZ. Šablona je v rutinách, které se submaticemi pracují, realizována polem indexů odpovídající délky. Pro submatice typu $\mathrm{A}_{11}$ je to 14 prvků, pro submatice typu $\mathrm{A}_{12}$ je to 27 prvků. Na začátku rutin se šablona inicializuje na hodnoty odpovídající skutečným pozicím nenulových prvků na prvním řádku submatice. Při zpracování řádku se šablona aktualizuje na hodnoty odpovídající skutečným pozicím nenulových prvků na dalším řádku. Aby se nemuselo ošetřovat zkrácení posledních řádků submatice, jejichž počet je dán poloviční šířkou pásu, ukládají se do datového souboru i tyto řádky v plné délce, tj. 14, resp. 27 prvků. Proto jsou o poloviční šířku pásu zvětšeny i vektory, které se při výpočtech s těmito submaticemi používají.

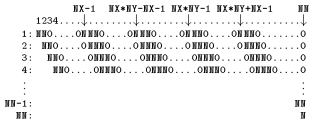
Kvůli zjednodušení transponování submatic typu $A_{12}$ jsou navíc ukládány v celé délce, tj. 27 prvků, i první řádky submatice, jejichž počet je dán rovněž poloviční šířkou pásu. Toto zvýšení paměťové náročnosti je stejně jako u původního formátu uložení matice tuhosti poměrně malé při současném výrazném zjednodušení algoritmů, které s maticí resp. jejími submaticemi pracují.



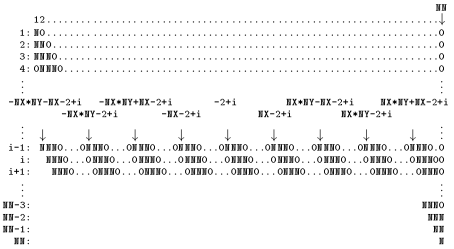

Obr. 5: Schéma blokového formátu uložení matice tuhosti

Uložení prvků jednotlivých submatic matice tuhosti v blokovém formátu vychází z uložení prvků matice tuhosti v původním formátu, z čehož vyplývá i jejich značná podobnost. Důsledkem toho jsou jejich shodné vlastnosti z hlediska paměťové náročnosti a složitosti algoritmů, které je používají. Výhoda a účel blokového formátu spočívá v rozdělení složek posunutí do bloků, což umožňuje efektivní paralelní implementaci metody sdružených gradientů. Výhodou původního formátu je menší počet vstupně/výstupních operací při srovnatelné sekvenční implementaci metody.

Podrobná zobrazení formátu uložení submatic jsou pro typ $A_{11}$ na obr. 6 a pro typ $A_{12}$ na obr. 7. Písmeno `N` v obrázcích označuje nenulový prvek.



Obr. 6: Blokový formát uložení submatice typu $A_{11}$



Obr. 7: Blokový formát uložení submatice typu $A_{12}$

Transformaci datových struktur úlohy, tj. matice tuhosti a vektoru pravých stran, z původního do blokového formátu řeší program DSPLIT (DATA SPLIT).

# 3 Paralelizace sdružených gradientů

Soustavu lineárních algebraických rovnic

$$Au = f$$

vznikající řešením 3D úloh pružnosti lze zapsat ve tvaru

$$\begin{aligned} A_{11}u_x + A_{12}u_y + A_{13}u_z &= f_x, \\ A_{12}^{\mathtt{T}}u_x + A_{22}u_y + A_{23}u_z &= f_y, \\ A_{13}^{\mathtt{T}}u_x + A_{23}^{\mathtt{T}}u_y + A_{33}u_z &= f_z, \end{aligned}$$

kde $u_x$, $u_y$, $u_z$ jsou vektory obsahující složky posunutí ve směrech x, y, z, $u = (u_x, u_y, u_z)$. Pokud pro zkrácení a zpřehlednění zvolíme

$$\begin{aligned} A_{11}u_x + A_{12}u_y + A_{13}u_z &= \hat{A}_x u, \\ A_{12}^{\mathtt{T}}u_x + A_{22}u_y + A_{23}u_z &= \hat{A}_y u, \\ A_{13}^{\mathtt{T}}u_x + A_{23}^{\mathtt{T}}u_y + A_{33}u_z &= \hat{A}_z u, \end{aligned}$$

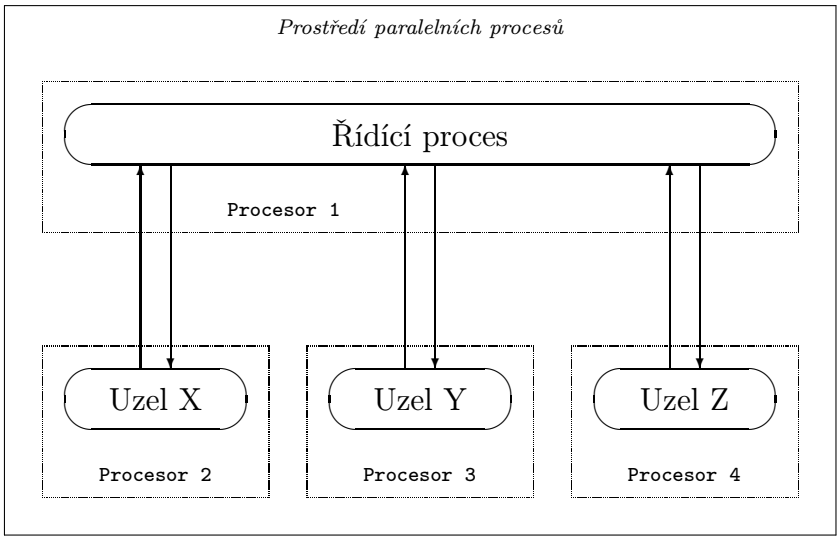
můžeme úlohu popsat soustavou

$$\begin{aligned} \hat{A}_x u &= f_x, \\ \hat{A}_y u &= f_y, \\ \hat{A}_z u &= f_z. \end{aligned}$$

V našem přístupu k paralelizaci metody sdružených gradientů použijeme datové struktury dané výše uvedeným rozdělením výchozí soustavy. Pro výpočet využijeme tři samostatné, ale vzájemně komunikující procesy, kterým budeme říkat uzly. Uzly běží současně každý na jednom procesoru, přičemž mezi nimi dochází k výměně informací potřebných pro jejich další běh. Algoritmicky jsou procesy uzlů identické, liší se pouze zpracovávanými daty v závislosti na tom, kterou složku posunutí řeší.

Pro vzájemnou synchronizaci uzlů je nezbytný další proces, kterému budeme říkat řídící. Řídící proces zabezpečuje spuštění uzlů, řídící a datovou

komunikaci mezi uzly a ukončení uzlů. Řídící komunikací rozumíme výměnu informací, které ovlivňují algoritmus metody, např. povel k provedení další iterace, potvrzení ukončení inicializace předpodmiňovače, apod.. Datovou komunikací rozumíme výměnu dat, např. koeficientu $a$, skalárního součinu $srr$, apod.. Řídící proces je algoritmicky jednoduchý, proto jej lze spustit na jednom procesoru společně s uzlem nebo na samostatném procesoru.




Obr. 8: Schéma paralelních procesů

Schéma paralelních procesů je na obr. 8. Označení uzlů je dáno složkou posunutí, kterou řeší. Každý proces běží na jednom procesoru, šipkami je v obrázku znázorněná komunikace mezi uzly a řídícím programem.

## 3.1 Algoritmus

Algoritmus paralelní verze metody sdružených gradientů s předpodmíněním je podobný sekvenční verzi, výpočet probíhá rovněž ve třech fázích. V levém sloupci jsou jednotlivé kroky algoritmu prováděné každým z uzlů, zde rozlišené indexem $i = x, y, z$, v pravém sloupci je výpočet řídícího procesu a prostřední sloupec zachycuje datovou komunikaci mezi uzly a řídícím procesem.

**I. Příprava** – nultá iterace (ITER=0), počáteční hodnoty proměnných.

$$
\begin{aligned}
r_i &\leftarrow f_i \\
sff_i &= < r_i, r_i > \\
u_i &\leftarrow u_{init,i} \\
w_i &\leftarrow \hat{A}_i u \\
re_i &\leftarrow r_i - w_i \\
v_i &\leftarrow C_i^{-1} re_i \\
srr0_i &= < re_i, re_i > \\
srw0_i &= < re_i, v_i >
\end{aligned}
$$

$sff_i,\ srr0_i,\ srw0_i \Rightarrow$

$$
\begin{aligned}
sff &= \sum sff_i \\
srr0 &= \sum srr0_i \\
srw0 &= \sum srw0_i
\end{aligned}
$$

**II. Opakovaný výpočet** – iterace (ITER=1,2,...).

$$
\begin{aligned}
w_i &\leftarrow \hat{A}_i v \\
s_i &= < w_i, v_i >
\end{aligned}
$$

$s_i \Rightarrow$

$$
\begin{aligned}
s &= \sum s_i \\
a &= srw0/s
\end{aligned}
$$

$\Leftarrow a$

$$
\begin{aligned}
u_i &\leftarrow u_i + a * v_i \\
re_i &\leftarrow re_i - a * w_i \\
w_i &\leftarrow C_i^{-1} re_i \\
srr_i &= < re_i, re_i > \\
srw_i &= < re_i, w_i >
\end{aligned}
$$

$srr_i,\ srw_i \Rightarrow$

$$
\begin{aligned}
srr &= \sum srr_i \\
srw &= \sum srw_i \\
b &= srw/srw0 \\
srw0 &= srw
\end{aligned}
$$

$\Leftarrow b$

$$
v_i \leftarrow w_i + b * v_i
$$

**III. Testování** – následuje po každé iteraci.

$$\begin{aligned} tol &= \sqrt{\tfrac{srr}{sff}} \\ tol &\leq \varepsilon \end{aligned}$$

V algoritmu není zobrazena řídící komunikace mezi uzly a řídícím procesem, která probíhá před přípravnou fází, kdy řídící proces pošle uzlům informaci o zahájení výpočtu, a před každou iterací, kdy řídící proces vypočítá dosaženou relativní přesnost. Na základě toho pošle uzlům buď informaci, aby pokračovali ve výpočtu další iterací (iterace musí být provedena všemi uzly nebo žádným) nebo informaci o ukončení výpočtu. V tom případě uzly uloží řešení do datových souborů a následně se ukončí.

## 3.2 Násobení matice krát vektor

V každé iteraci se provádí operace násobení matice krát vektor

$$w_i \leftarrow \hat{A}_i v,$$

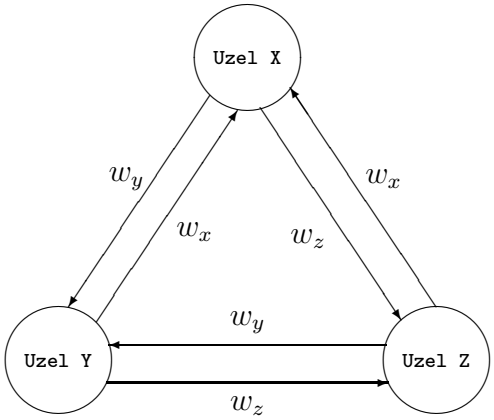
což přepsáno znamená pro každý uzel jednu z rovnic

$$\begin{aligned} w_x &= A_{11}v_x + A_{12}v_y + A_{13}v_z, \\ w_y &= A_{12}^{\mathtt{T}}v_x + A_{22}v_y + A_{23}v_z, \\ w_z &= A_{13}^{\mathtt{T}}v_x + A_{23}^{\mathtt{T}}v_y + A_{33}v_z. \end{aligned}$$

Z toho vyplývá, že si uzly mezi sebou v každé iteraci musí posílat vektory. V $i$-tém uzlu se provádí výpočet se čtyřmi vektory, a to buď s $w_i$, $v_x$, $v_y$, $v_z$ nebo s $v_i$, $w_x$, $w_y$, $w_z$. Budeme uvažovat druhý případ:

| Uzel X: | | | Uzel Y: | | | Uzel Z: | | |
|---|---|---|---|---|---|---|---|---|
| $w_x$ | = | $A_{11}v_x$ | $w_y$ | = | $A_{22}v_y$ | $w_z$ | = | $A_{33}v_z$ |
| $w_y$ | = | $A_{12}^{\mathtt{T}}v_x$ | $w_z$ | = | $A_{23}^{\mathtt{T}}v_y$ | $w_x$ | = | $A_{13}v_z$ |
| $w_z$ | = | $A_{13}^{\mathtt{T}}v_x$ | $w_x$ | = | $A_{12}v_y$ | $w_y$ | = | $A_{23}v_z$ |

Po tomto výpočtu dojde k datové komunikaci mezi uzly, při které si vzájemně posílají vektory. V $i$-tém uzlu zůstává vektor $w_i$, vektory $w_j$, $j \neq i$ se podle indexu $j$ pošlou ostatním uzlům, které je přičtou k vlastnímu vektoru $w_j$. Tato situace je zachycena na obr. 9.




Obr. 9: Výměna vektorů mezi uzly při násobení matice krát vektor

Při datové komunikaci $i$-tý uzel nejprve vektory $w_j$, $j \neq i$ odešle, a teprve potom přijímá od ostatních uzlů vektory – příspěvky, které přičte k vektoru $w_i$.

Při komunikaci uzlů může dojít k porušení jejich vzájemné synchronizace. Tomu jde zabránit, pokud datová komunikace mezi uzly bude probíhat prostřednictvím řídícího procesu. Řídící proces nejprve přijme oba dva vektory poslané každým uzlem a uloží je do datového bufferu v pořadí, ve kterém je bude zpětně odesílat uzlům. Teprve po přijetí všech vektorů je řídící proces odešle zpět uzlům. Tímto způsobem se zabrání možné kolizi.

## 3.3 Předpodmínění

Předpodmínění je implementováno stejným způsobem jako u sekvenční verze, tedy modifikací residua

$$w_i = C_i^{-1} re_i,$$

tj. řešením soustavy

$$C_i w_i = r e_i$$

v každé iteraci. Předpodmiňující matice $C_i$ je pro každý uzel jiná. Budeme uvažovat stejné typy předpodmínění jako u sekvenční verze, volby matic $C_x$, $C_y$, $C_z$ budou následující:

- $C_x = C_y = C_z = I$,

  kde $I$ je jednotková matice, výpočet probíhá bez předpodmínění,

- $C_x = D_x$, $C_y = D_y$, $C_z = D_z$,

  kde $D_x$ je diagonální část matice $A_{11}$, $D_y$ je diagonální část matice $A_{22}$, $D_z$ je diagonální část matice $A_{33}$, potom hovoříme o diagonálním předpodmínění,

- $C_x = (X_x - L_x)X_x^{-1}(X_x - L_x)^{\mathtt{T}}$,

  $C_y = (X_y - L_y)X_y^{-1}(X_y - L_y)^{\mathtt{T}}$,

  $C_z = (X_z - L_z)X_z^{-1}(X_z - L_z)^{\mathtt{T}}$,

  kde $L_x$ je dolní trojúhelníková část matice $A_{11}$, $X_x$ je diagonální matice určená podmínkou $C_x e = A_{11} e$, $L_y$ je dolní trojúhelníková část matice $A_{22}$, $X_y$ je diagonální matice určená podmínkou $C_y e = A_{22} e$, $L_z$ je dolní trojúhelníková část matice $A_{33}$, $X_z$ je diagonální matice určená podmínkou $C_z e = A_{33} e$, $e = (1, \ldots, 1)^{\mathtt{T}}$, potom hovoříme o předpodmínění DD-IF.

Typ předpodmínění je pro všechny uzly stejný, přestože každý uzel provádí jeho výpočet nezávisle na ostatních uzlech.

## 3.4 Počáteční aproximace

Rovněž počáteční aproximace $u_{init,i}$ je implementována stejným způsobem jako u sekvenční verze. U jednotlivých uzlů se výpočet počáteční aproximace liší. Uvažujme tyto volby počáteční aproximace $u_{init,x}$, $u_{init,y}$, $u_{init,z}$:

- $u_{init,x} = u_{init,y} = u_{init,z} = 0$,

  potom hovoříme o nulové počáteční aproximaci,

- $u_{init,x} = C_x^{-1} r_x,\ u_{init,y} = C_y^{-1} r_y,\ u_{init,x} = C_x^{-1} r_x$,

  pro metodu sdružených gradientů s předpodmíněním, $C_x$, $C_y$, $C_z$ jsou předpodmiňující matice dané typem předpodmínění pro jednotlivé uzly, $r_x$, $r_y$, $r_z$ je pravá strana soustavy $r$ rozdělená podle složek posunutí,

- $u_{init,x} \leftarrow \tilde{u}_{init,x},\ u_{init,y} \leftarrow \tilde{u}_{init,y},\ u_{init,z} \leftarrow \tilde{u}_{init,z}$,

  pro načtení počáteční aproximace z datových souborů, přičemž $\tilde{u}_{init,x}$, $\tilde{u}_{init,y}$, $\tilde{u}_{init,z}$ je výsledek dříve provedeného výpočtu rozděleného podle složek posunutí ($\tilde{u}_{init,x}$, $\tilde{u}_{init,y}$, $\tilde{u}_{init,z}$ můžou být například produkty jiné numerické metody aplikované na stejnou úlohu).

- $u_{init,x} \leftarrow u_x,\ u_{init,y} \leftarrow u_y,\ u_{init,z} \leftarrow u_z$,

  pro tuto volbu počáteční aproximace nazvanou „částečná řešení“, bude paralelní implementace značně efektivní, neboť výpočty řešení soustav

$$\begin{aligned} A_{11}\ u_x &= r_x, \\ A_{22}\ u_y &= r_y, \\ A_{33}\ u_z &= r_z, \end{aligned}$$

  metodou sdružených gradientů s diagonálním předpodmíněním mohou běžet současně. Z důvodu správné synchronizace uzlů je důležité, aby proběhla řídící komunikace mezi každým uzlem a řídícím procesem poté, co uzly ukončí počáteční aproximaci $u_{init,x}$, $u_{init,y}$, $u_{init,z}$ (uzly pošlou řídícímu procesu potvrzení ukončení výpočtu počáteční aproximace).

Typ počáteční aproximace je pro všechny uzly stejný, přestože každý uzel provádí její výpočet nezávisle na ostatních uzlech.

# 4 Systémové prostředky

Systémové prostředky, které byly užívány při realizaci diplomové práce, představují především tři pracovní stanice IBM RS/6000, na kterých byl vytvořen virtuální stroj pracující pod prostředím PVM. Popis těchto prostředků je v následujících oddílech.

Sekvenční verze řešiče byla vytvořena a odladěna na počítači typu PC, na kterém byly rovněž provedeny některé testy:

| | |
|---|---|
| Procesor: | Cyrix 486 |
| Taktovací frekvence: | 66 MHz |
| Operační paměť: | 4 MB |
| Vnější paměť: | 210 MB |
| Operační systém: | MS DOS verze 6.22 |
| Překladač Fortranu: | MS Fortran verze 4.10 |

Nevhodná architektura počítače především z hlediska organizace operační paměti, výkon počítače a operační systém dovolují použít daný počítač pouze k řešení soustav lineárních rovnic velmi malého rozsahu ($10^2$ - $10^3$ rovnic). Takové soustavy se řeší při použití počítače této třídy přímými metodami.

## 4.1 Stanice IBM RS/6000

Pro řešení rozsáhlých soustav lineárních rovnic lze použít pracovní stanice IBM RS/6000, které jsou koncipovány pro výpočetně náročné aplikace. Stejné označení všech počítačů neznamená, že jsou výkonově srovnatelné, ale že jsou do značné míry binárně kompatibilní. Základem každého počítače je jiný RISC-ový procesor firmy IBM, proto jsou jednotlivé počítače výkonově značně rozdílné.

Pracovní stanice IBM RS/6000 pracují pod operačním systémem IBM AIX verze 3.2, který je verzí operačního systému Unix rozšířeného o funkce pro snadnou údržbu systému a zvyšující uživatelský komfort. Pro překlad zdrojových textů napsaných v jazyku Fortran 77 je určen překladač AIX XL-Fortran verze 2.3.

Při realizaci programové části diplomové práce byla používána skupina tří pracovních stanic IBM RS/6000, které pracovaly v jedné lokální síti. Skupině spřažených pracovních stanic se říká klastr (cluster) nebo farma. Na

klastr lze pohlížet jako na jeden paralelní počítač s distribuovanou pamětí, na kterém mohou být spuštěny paralelní programy. Pro komunikaci mezi procesy, které běží na jednotlivých procesorech, se využívá lokální síť. Toto řešení je pomalejší než u výkonově srovnatelného multiprocesorového počítače s distribuovanou pamětí, který prostředky pro komunikaci úloh na jednotlivých procesorech obsahuje. Pokud objem přenášených dat při komunikaci paralelních procesů není velký, výkon multiprocesorového počítače a klastru pracovních stanic je srovnatelný. U multiprocesorového počítače jsou stejně výkonné jednotlivé procesory, u skupiny pracovních stanic, která se skládá z různých počítačů (jako v našem případě, viz. následující oddíly), jsou výkony jednotlivých procesorů rozdílné.

### 4.1.1 Model 320

Prvním počítačem řady 300 je model 320 s následujícími charakteristikami:

| | |
|---|---|
| Procesor: | Power |
| Taktovací frekvence: | 20 MHz |
| Operační paměť: | 24 MB |
| Vnější paměť: | 1,5 GB |

Procesor Power je typickým představitelem RISC-ové architektury. Jednotlivé funkční bloky procesoru pracují paralelně, což umožňuje vykonávání více instrukcí během jednoho strojového cyklu.

V současné době je celková koncepce tohoto počítače zastaralá, neboť počítač byl uveden do provozu před více než pěti lety a firma IBM za tu dobu vyvinula nové technologie, které zvyšují výkon procesoru nebo počítače. Tento výkonově nejslabší počítač bude v testech označen jako `prokop`.

### 4.1.2 Model 3CT

Posledním počítačem řady 300 je model 3CT, který má tyto charakteristiky:

| | |
|---|---|
| Procesor: | Power2 |
| Taktovací frekvence: | 67 MHz |
| Operační paměť: | 128 MB |
| Vnější paměť: | 2 GB |

Procesor Power2 je další „generací“ procesoru Power. Použitá technologie zvyšuje počet instrukcí, které je procesor schopen provádět během jednoho strojového cyklu, čímž značně roste výkon počítače.

Výkonově se tento počítač řadí na první místo, ve srovnání s modelem 320 je přibližně sedminásobně rychlejší. V testech bude označen jako `fany`.

### 4.1.3 Model 250

Počítače řady 200 jsou postaveny na procesoru PowerPC. Jeden z posledních počítačů řady 200 je model 250 s následujícími charakteristikami:

| | |
|---|---|
| Procesor: | PowerPC |
| Taktovací frekvence: | 66 MHz |
| Operační paměť: | 16 MB |
| Vnější paměť: | 1 GB |

Procesor PowerPC je první z procesorů, které byly vyvinuty při společném projektu firem IBM, Apple a Motorola. Počítače firmy IBM postavené s použitím procesoru PowerPC dosahují vysokého výkonu zejména zlepšením technologie pro přístup procesoru do paměti a použitím rychlé sběrnice.

Výkonově se tento počítač řadí mezi model 3CT a 320, je přibližně trojnásobně rychlejší než model 320, má však menší operační paměť. V testech bude označen jako `geam`.

## 4.2 Prostředí PVM

PVM je zkratka pro Parallel Virtual Machine. Je to balík softwaru, který umožňuje, aby heterogenní síť vytvořená ze sekvenčních a paralelních počítačů vypadala jako jeden paralelní výpočetní prostředek. PVM vytváří jednotné prostředí na různých počítačích pracujících pod operačním systémem Unix, které umožňuje běh paralelních aplikací. Rozsahem i výpočetně náročné problémové úlohy tak mohou být řešeny rozložením práce mezi počítače, které pracují pod PVM. Podrobnější informace o tomto prostředí lze nalézt v [7].

Vývoj prostředí PVM začal v létě roku 1989 v Oak Ridge National Laboratory ve státě Tennessee v USA. Postupně se k tomuto projektu přidaly další instituce, mezi nimi univerzita v Tennessee, superpočítačové centrum v Pittsburghu a další. Software PVM je v současné době volně šiřitelný mezi

zájemce, zejména z řad vědeckovýzkumných pracovníků, proto vznikají po celém světě aplikace, které jej využívají. Problémy, připomínky a podněty uživatelé konzultují s autory, čímž ovlivňují budoucí verze PVM a pomáhají při rozvoji tohoto projektu.

Pod prostředím PVM pracuje uživatelsky definovaná skupina sériových a paralelních počítačů, která se jeví jako jeden velký paralelní počítač s distribuovanou pamětí neboli paralelní virtuální stroj (parallel virtual machine). Odtud také vzniklo označení prostředí. Počítač, na kterém mohou běžet úlohy, se nazývá hostitelský (host). PVM obsahuje prostředky pro automatické spouštění procesů, přičemž podporuje jejich vzájemnou komunikaci a synchronizaci. Takové procesy, které spolu komunikují a mohou pracovat při řešení problémů paralelně, se nazývají úlohy (tasks).

Virtuální stroj umí PVM vytvořit i na heterogenní síti. Při výpočtu mohou spolupracovat počítače naprosto odlišných architektur, s naprosto odlišnými výpočetními možnostmi a kapacitami, sériové, paralelní i vektorové. Heterogenní mohou být i technologie jejich vzájemného propojení. Počítače mohou být provázány sítěmi typu Ethernet, FDDI, Token Ring, .... Díky značné různorodosti hardwaru nemůže PVM pro vzájemnou komunikaci úloh využít rozhraní Unix sockets nebo TCP/IP, pro komunikaci si tedy vytváří vlastní prostředky. Různými architekturami jsou data interpretována různými způsoby, proto PVM zajišťuje konverzi dat při komunikaci. Díky těmto vlastnostem PVM mohou při řešení problémů spolupracovat počítače DEC Alpha, Cray, NeXT, Intel Paragon, IBM RS/6000, Silicon Graphics, Sun 4 a další. Přitom počet strojů, které PVM podporuje, neustále roste.

PVM se skládá ze dvou částí. První část tvoří démon `pvmd3`. Ten se stará o zajištění služeb PVM, tzn. spouštění, synchronizaci, ukončení úloh a jejich vzájemnou komunikaci. Spuštěním `pvmd3` se na všech uživatelsky definovaných hostitelích spustí lokální démon `pvmd3`, který zajišťuje na daném hostiteli funkce PVM a který naváže spojení se všemi lokálními démony `pvmd3` na ostatních hostitelích. Tím je vytvořen virtuální stroj pracující pod PVM, který je připraven ke spouštění aplikací. Druhou část tvoří knihovny funkcí PVM, které aplikacím slouží ke komunikaci s lokálním démonem `pvmd3`, a tím využívají služeb PVM. Tedy dvě úlohy, které spolu komunikují prostřednictvím volání knihovních funkcí, ve skutečnosti komunikují pouze se svými lokálními démony `pvmd3` na hostitelích a démony komunikují mezi sebou. Výjimku tvoří dynamické skupiny úloh. Od verze 3 PVM podporuje práci s celou skupinou úloh najednou, přičemž se složení této skupiny může

dynamicky v čase měnit a naopak úloha může patřit do několika skupin současně. V rámci jedné skupiny se mohou provádět některé operace, například posílání dat. Aplikace, jejíž úlohy pracují s dynamickými skupinami úloh, musí volat funkce speciální knihovny. Protože služby pro dynamické skupiny úloh nejsou obsaženy v démonu `pvmd3`, je při prvním volání funkce této speciální knihovny spuštěn server, který tyto služby obstará.

### 4.2.1 Spuštění a konfigurace

Vytvořit virtuální stroj neboli spustit prostředí PVM lze dvěma způsoby: z příkazové řádky se zadá `pvmd3` nebo `pvm`. První příkaz spouští pouze démon, druhý navíc ještě uživatelskou konzolu, která umožňuje interaktivní ovládání PVM, tzn. spouštění, ukončování úloh, zjištění informací o stavu úloh a hostitelů. Navíc lze z konzoly dynamicky měnit množinu hostitelů nebo jejich nastavení, tedy konfigurovat virtuální stroj.

Druhý způsob, kterým lze ovlivnit počáteční nastavení virtuálního stroje, je spuštění prostředí PVM s parametrem, kterým je jméno konfiguračního souboru. Pravidla pro vytvoření tohoto souboru jsou stanovená v [7].

Příkladem takového souboru může být následující výpis:

```
# Komentářový řádek

geam wd=/u/bigdisk ep=/u/bigdisk
fany wd=/u/bigdisk ep=/u/bigdisk

* wd=/u/local
```

Obr. 10: Konfigurační soubor pro virtuální stroj

Do prostředí PVM jsou připojeny počítače `geam` a `fany`. Parametr `wd` nastavuje pracovní adresář a parametr `ep` nastavuje cestu ke spustitelnému tvaru aplikace na daném hostiteli. Pro všechny ostatní počítače pracující pod PVM bude platit implicitní nastavení, které je na řádku označeném $*$.

### 4.2.2 Tvorba aplikací

Tvorba paralelních aplikací pod PVM se odvíjí od typu řešeného problému, ale algoritmizace problému je vždy založena na komunikaci úloh při výpočtech, které vedou ke společnému vyřešení problému. Pro paralelní aplikace založené na posílání zpráv jsou typické dva přístupy: doménová dekompozice a funkční dekompozice.

Při prvním přístupu se aplikace skládá z množiny identických, rovnocenných úloh, které jsou spuštěny současně, přičemž každá řeší část původního problému. Podproblémy jsou zpravidla stejně rozsáhlé a výpočetně náročné pro všechny úlohy. Fyzicky na disku existuje pouze jediný spustitelný tvar jedné úlohy, v případě spuštění aplikace se na virtuálním stroji automaticky spustí příslušný počet kopií této úlohy.

Druhý přístup zahrnuje ty aplikace, v nichž skupina řízených úloh pracuje pro jednu nebo více řídících úloh. Řídící úloha je ve většině aplikací jedna. Řízené úlohy mohou být stejné, ale nemusí to být pravidlem. Jako příklad poslouží možná reálná situace, kdy virtuální stroj je složen z vektorového počítače, dvou sériových počítačů a grafické stanice. Aplikace se skládá z jedné řídící úlohy, která běží na sériovém počítači, a více řízených úloh. Úloha, která běží na vektorovém počítači, se stará o výpočty, při kterých se pracuje s vektory. Další úloha běžící na druhém sériovém stroji zvládá všechny ostatní výpočty a na grafické stanici třetí úloha zobrazuje výsledky.

Uživatelé PVM mohou své aplikace psát v programovacím jazyce C nebo ve Fortranu, překlad zdrojového kódu do spustitelného tvaru probíhá podle stejných pravidel jako v případě běžných aplikací ve dvou fázích: kompilace a spojení.

Pokud programátor použije ve zdrojovém souboru předdefinované konstanty při volání funkcí PVM, musí být zdrojové texty v jazyce C kompilovány spolu se souborem `pvm3.h` a zdrojové texty napsané v jazyce Fortran kompilovány se souborem `fpvm3.h`. V opačném případě není nutné uvedené soubory použít.

Aplikace musí být spojovány s knihovnami funkcí PVM. Pro jazyk C se knihovní soubor jmenuje `libpvm3.a` a pro jazyk Fortran jsou knihovní soubory dva, `libfpvm3.a` a `libpvm3.a`, přičemž pořadí spojování knihoven musí být zachováno. Pokud aplikace používá dynamické skupiny úloh, potom aplikace v jazyce C musí být spojovány se soubory `libgpvm3.a`, `libpvm3.a` a v jazyce Fortran se soubory `libfpvm3.a`, `libgpvm3.a`, `libpvm3.a` vždy

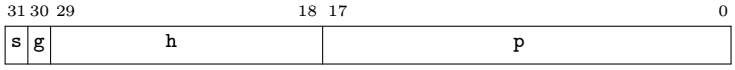
v uvedeném pořadí.

V souvislosti s překladem bych se chtěl zmínit o možných problémech s hlavičkovými soubory. Po překladu zdrojových textů řešičů napsaných v jazyce Fortran na pracovní stanici IBM RS/6000 mi aplikace havarovala při předávání parametrů podprogramům, případně byly tyto hodnoty nulové nebo se vyskytovaly jiné chyby. Problém byl v překladu souboru `fpvm3.h`. Překladač celočíselným konstantám deklarovaným příkazem `PARAMETER` vyhradil dva bajty paměti místo čtyř, které jim vyhradil v programu v místech jejich použití.

### 4.2.3 Identifikace hostitelů a úloh

K identifikaci hostitelů a úloh slouží čtyřbajtové celé číslo, které je obdobou identifikačních čísel v operačním systému Unix. Na existenci identifikačních čísel je v PVM založena veškerá komunikace, neboť identifikační číslo představuje v rámci virtuálního stroje buď adresu úlohy nebo adresu lokálního démonu `pvmd3`, tzn. adresu hostitele. V dalším textu bude identifikační číslo hostitele označeno `hostid` a identifikační číslo úlohy `taskid`.

Identifikační číslo se skládá ze čtyř částí, viz. obr. 11. Kombinace bitů




Obr. 11: Struktura identifikačního čísla

`s` a `g` určuje typ identifikačního čísla. Platí: `s=0`, `g=0` označuje identifikační číslo úlohy a `s=1`, `g=0` označuje identifikační číslo lokálního démonu `pvmd3` resp. označuje hostitele. Pokud budou oba bity jedničkové, identifikační číslo je chybné.

Pole dvanácti bitů `h` určuje číslo hostitele. Pokud je ve virtuálním stroji zapojeno `numhost` počítačů, rozsah hodnot `h` je `1..numhost`. Z toho vyplývá, že maximální počet počítačů vytvářejících jeden virtuální stroj je $2^{12} - 1 = 4095$.

Poslední pole o velikosti osmnácti bitů určuje číslo úlohy. Pokud je na virtuálním stroji spuštěno `numtasks` úloh, rozsah hodnot `p` je `1..numtasks`.

Maximální počet úloh spuštěných na jednom virtuálním stroji je $2^{18} - 1 = 262143$.

Hostitel, který byl do PVM připojen jako čtvrtý, bude mít `hostid` rozepsané podle jednotlivých polí 1-0-4-0. Pokud na tomto hostiteli poběží dvanáct úloh, bude `taskid` poslední spuštěné úlohy 0-0-4-12.

### 4.2.4 Vytvoření úlohy

Aby spuštěná úloha mohla využívat prostředí PVM, musí komunikovat s lokálním démonem `pvmd3`, který se nachází na stejném hostiteli jako běžící úloha. K tomu účelu musí použít volání funkcí, které se nacházejí v knihovních souborech (viz. oddíl Tvorba aplikací). Jaké funkce použije, to si ukážeme v tomto a následujících oddílech. Použití funkcí bude uvedeno na příkladech v jazyku Fortran, ve kterém jsou napsány všechny zdrojové texty přiložené k diplomové práci.

```
PROGRAM master

INTEGER*4 mytaskid, taskids(3)
INTEGER*4 ntasks, info                                  PROGRAM slave
CHARACTER node*5
                                                        INTEGER*4 mytaskid, info
node = 'slave'                                          INTEGER*4 mtaskid

CALL pvmfmytid( mytaskid )                              CALL pvmfmytid( mytaskid )

CALL pvmfspawn( node, 0, '*', 3, taskids, ntasks )      CALL pvmfparent( mtaskid )

⋮                                                       ⋮


CALL pvmfexit( info )                                   CALL pvmfexit( info )

STOP                                                    STOP
END                                                     END
```

Obr. 12: Příklad paralelních procesů

Na začátku každé úlohy musí být volána funkce `pvmfmytid()` a před ukončením úlohy funkce `pvmfexit()`. Funkce `pvmfmytid()` registruje úlohu v PVM a vygeneruje její jedinečné `taskid`. Tato funkce musí být volána

před všemi ostatními funkcemi z knihovny PVM. Funkce `pvmfexit()` ukončuje komunikaci s lokálním démonem `pvmd3`, ale neukončuje úlohu. Proto je vhodné tuto funkci volat těsně před ukončením úlohy.

Důležitá z hlediska tvorby aplikací je funkce `pvmfspawn()`, která na virtuálním stroji spustí zadaný počet kopií zadaného programu. Funkce nabízí možnost ovlivnění, na kterém hostiteli se kopie programu budou spouštět. Lze explicitně zadat konkrétního hostitele nebo typ architektury hostitele. Nechybí ani možnost nechat spuštění kopií programu na PVM, které je rozdělí mezi hostitele automaticky. Funkce vrací počet skutečně spuštěných kopií programu a množinu jejich `taskid`. Úlohy, které vznikly pomocí funkce `pvmfspawn()`, mohou zjistit `taskid` této „rodičovské“ úlohy. K tomu účelu slouží funkce `pvmfparent()`.

Na obr. 12 jsou uvedeny zdrojové texty programů, které demonstrují použití výše uvedených funkcí. Po překladu se na virtuálním stroji spustí program master (řídící proces), který spustí další tři kopie programu slave (řízený proces). Tímto způsobem budou na virtuálním stroji běžet čtyři úlohy současně. Tyto úlohy jsou již připraveny ke komunikaci, která může probíhat mezi řídícím a řízenými procesy.

### 4.2.5 Komunikace úloh

Komunikace mezi úlohami v prostředí PVM je založena na modelu posílání zpráv. Zprávou rozumíme skupinu dat, kterou jedna úloha posílá druhé. Zpráva se nepředává přímo, odesílající úloha ji uloží do vyrovnávací paměti v prostředí PVM, který se nazývá buffer zpráv. Odtud si ji vyzvedne ta úloha, která je příjemce zprávy.

Buffer zpráv neexistuje v PVM pouze jediný, úlohy mohou množinu bufferů zpráv dynamicky v čase měnit. Počet ani velikost bufferů zpráv není omezena. Každá úloha může pracovat s více buffery zpráv najednou, ale okamžitě přístupný je pro ni pouze jeden pro čtení zpráv a jeden pro zápis zpráv. Pro přístup k bufferům zpráv se používají stejná identifikační čísla jako pro hostitele nebo pro úlohy.

Součástí zprávy jsou `taskid` odesílající i přijímací úlohy a třída zprávy. Třída zprávy je uživatelsky definované, celé, čtyřbajtové číslo, které slouží k rozlišení zpráv. Toto rozlišení se může použít v případě rozdílné struktury posílaných dat. Dalším případem, kde se používá inkrementace třídy zpráv, je aplikace, kdy zprávy většinou putují mezi úlohami jedním směrem. Pokud

by byla třída zpráv pro všechny zprávy stejná a úloha, která má zprávy přijmout, je nestačí zpracovávat, mohlo by nastat situace, kdy v bufferu zpráv nová zpráva přemaže starou, což by mohlo ovlivnit výpočet.

Procedura, ve které ukládá odesílající úloha zprávu do bufferu zpráv, probíhá ve třech krocích. Nejprve musí být patřičný buffer zpráv inicializován, k tomu slouží funkce `pvmfinitsend()`. V dalším kroku se opakovaně volá funkce `pvmfpack()`, která „balí“ data a ukládá je do bufferu zpráv. Nakonec se volá funkce `pvmfsend()`, která je odešle. V případě posílání téže zprávy několika úlohám současně, lze použít funkci `pvmfmcast()`.

```
INTEGER*4 msg, n                                INTEGER*4 msg, n
REAL*8 data(100), res3                          REAL*8 data, res
⋮                                               ⋮

msg = 1                                         msg = 1
CALL pvmfinitsend( 0, info )                    CALL pvmfrecv( mtaskid, msg, info )
CALL pvmfpack( INTEGER4, n, 1, 1, info )        CALL pvmfunpack( INTEGER4, n, 1, 1, info )
CALL pvmfpack( REAL8, data, n, 1, info )        CALL pvmfunpack( REAL8, data, n, 1, info )
CALL pvmfmcast( 3, taskids, msg, info )
⋮                                               ⋮

⋮                                               ⋮

msg = 2                                         msg = 2
CALL pvmfrecv( taskids(3), msg, info )          CALL pvmfinitsend( 0, info )
CALL pvmfunpack( REAL8, res3, 1, 1, info )      CALL pvmfpack( REAL8, res, 1, 1, info )
                                                CALL pvmfsend( mtaskid, msg, info )
⋮                                               ⋮
```

Obr. 13: Příklad komunikace paralelních procesů

Příjem zprávy probíhá ve dvou krocích. V prvním kroku úloha přijme zprávu, která má odpovídající třídu zprávy a která má odpovídající `taskid` odesílající úlohy. Je možné nastavit, aby úloha přijímala zprávy všech tříd, zprávy od libovolné jiné úlohy nebo všechny zprávy. Příjem zprávy může být blokující nebo neblokující. Blokující příjem charakterizuje situaci, kdy úloha při volání funkce pro příjem zprávy čeká tak dlouho, dokud nějakou zprá-

vu nepřijme. Blokující příjem zprávy zajistí funkce `pvmfrecv()`. Neblokující příjem zprávy charakterizuje situaci, kdy úloha při volání funkce pro příjem zprávy zjistí, zda se pro ni v bufferu zpráv nachází nějaká zpráva. Pokud ano, tak ji přijme, v opačném případě nečeká, dokud ji taková zpráva nepřijde , ale pokračuje dále ve výpočtu. Neblokující příjem zprávy obstará funkce `pvmfnrecv()`. Je zřejmé, že po volání funkce pro neblokující příjem zprávy je nutné testovat, zda byla nebo nebyla zpráva přijata. Druhý krok přijetí zprávy spočívá v „rozbalení“ přijatých dat. Tuto činnost, která se provádí opakovaně, se provede voláním funkce `pvmfunpack()`.

Příklad na obr. 13 ukazuje komunikaci paralelních úloh, které jsme vytvořili v minulém oddíle. Na levé straně je zdrojový text pro program master, na pravé straně zdrojový text pro program slave. První zprávu posílá řídící úloha (úloha master) všem řízeným úlohám (tři úlohy slave). Struktura zprávy se skládá z jednoho celého čísla `n`, které udává počet prvků pole reálných čísel, které bude následovat. Maximální velikost `n` je `100`. Druhou zprávu posílají řízené úlohy řídící, poté co z poslaných dat spočítají výsledek `res`. Řídící program ale přijímá pouze zprávu od třetí kopie programu slave, ostatní jsou ignorovány. Zpráva obsahuje pouze jedno reálné číslo představující výsledek výpočtu provedeného třetí kopií programu slave.

### 4.2.6 Další funkce

Další funkce, které úlohy v prostředí PVM mohou používat, se týkají dynamické konfigurace virtuálního stroje. Nastavení virtuálního stroje, která lze ovlivňovat při startu PVM prostřednictvím konfiguračního souboru nebo interaktivně z uživatelské konzoly (viz. oddíl Spuštění a konfigurace), mohou ovlivnit také úlohy, které na tomto virtuálním stroji běží. Přitom možnosti všech uvedených způsobů konfigurace virtuálního stroje jsou srovnatelné.

PVM obsahuje také funkce, které rozšiřují možnosti komunikace úloh. Úlohy mohou vzájemně komunikovat prostřednictvím tzv. signálů, které známe ze světa Unixu. Úlohy, které běží na virtuálním stroji pod prostředím PVM, mohou tímto způsobem komunikovat s úlohami, které běží na některém z hostitelů. Další možností komunikace úloh je generování zpráv při výskytu události. Událost může být např. ukončení úlohy, tzn. v případě, že úloha je ukončena, pošle předtím zprávu určené skupině úloh. Takový způsob posílání zpráv je asynchronní, protože úlohy, které mají zprávu přijmout, neví, kdy k takové události dojde. Prostředí PVM zahrnuje rovněž funkce pro

příjem těchto událostmi generovaných, asynchronních zpráv.

Největší podmnožina funkcí PVM jsou informační funkce, které mapují aktuální stav PVM. Patří mezi ně např. funkce `pvmftidtohost()`, která zjistí `hostid` hostitele, na kterém běží úloha se zadaným `taskid`, nebo funkce `pvmfhostsync()`, která přečte na zadaném hostiteli systémový čas.

# 5 Programová realizace

Součástí diplomové práce je disketa se zdrojovými texty programů DSPLIT, PCG–2 a PCG–3 napsanými v programovacím jazyku FORTRAN (specifikace 77). Tyto programy jsou určeny a odladěny zejména pro použití na pracovních stanicích IBM RS/6000 (zvolíme označení platformy pracovních stanic WS = workstation). Zdrojové texty programů jsou okomentovány, ale pro jejich větší rozsah nejsou uvededeny.

## 5.1 Konverzní program DSPLIT

DSPLIT je konverzní program, který převádí soustavu lineárních rovnic uloženou v původním, kompaktním tvaru do blokového formátu, ve kterém jsou odděleny složky posunutí. Tento formát uložení soustavy do datových souborů je důležitý pro efektivní paralelní implementaci metody sdružených gradientů s předpodmíněním (popis obou formátů uložení matice tuhosti je v kapitole Sdružené gradienty).

V původním formátu je soustava lineárních rovnic uložena v těchto třech datových souborech:

| | | |
|---|---|---|
| `FKBC.G32` | ... | matice tuhosti $A$, |
| `FRBC.G32` | ... | pravá strana $f$, |
| `FV.G32` | ... | údaje popisující úlohu. |

Po konverzi do blokového formátu jsou datové soubory, ve kterých je soustava uložena, následující:

| | | |
|---|---|---|
| `FKBCXX.G32` | ... | matice $A_{11}$, |
| `FKBCXY.G32` | ... | matice $A_{12}$, |
| `FKBCXZ.G32` | ... | matice $A_{13}$, |
| `FKBCYY.G32` | ... | matice $A_{22}$, |
| `FKBCYZ.G32` | ... | matice $A_{23}$, |
| `FKBCZZ.G32` | ... | matice $A_{33}$, |
| `FRBCX.G32` | ... | pravá strana $f_x$, |
| `FRBCY.G32` | ... | pravá strana $f_y$, |
| `FRBCZ.G32` | ... | pravá strana $f_z$, |
| `FV.G32` | ... | údaje popisující úlohu (beze změn). |

Tento program je odladěn a vyzkoušen na počítačích typu PC a na pracovních stanicích IBM RS/6000. Na disketě se v adresáři `DATA` nacházejí tyto soubory:

◇ `DSPLIT.FOR`,

zdrojový text hlavního programu,

◇ `READFV.FOR`,

zdrojový text rutiny pro načtení údajů ze souboru `FV.G32`, které popisují úlohu, tato rutina je univerzální pro všechny programy,

◇ `DSPLIT.BAT`,

dávkový soubor pro překlad programu na platformě PC,

◇ `DSPLIT.LNK`,

pomocný soubor pro překlad programu na platformě PC,

◇ `DSPLIT.WS`,

dávkový soubor pro překlad programu na platformě WS.

Jednotlivé kroky, které program po spuštění vykoná, jsou slovně popsány v podobě tohoto algoritmu:

*Začátek programu*

- Ze souboru `FV.G32` jsou načteny údaje, které popisují úlohu (důležité jsou parametry diskretizační sítě, pomocí které soustava vznikla). Tuto činnost obstará rutina `READFV`.

*Konverze matice tuhosti*

- Inicializace proměnných, nulování kruhových bufferů a inicializace šablony. Protože při přečtení jednoho řádku matice tuhosti v původním formátu jsou známy prvky, které se v blokovém formátu ukládají do datových souborů matic $A_{12}$, $A_{13}$, $A_{23}$ v několika dalších řádcích, ukládají se do třech kruhových bufferů. Pozice těchto prvků v bufferech

jsou uloženy v poli indexů, kterému se říká šablona. Šablona se na začátku programu inicializuje na odpovídající hodnoty a pro každý řádek matice se aktualizuje.

- Otevření příslušných datových souborů.
- Cyklus přes všechny řádky matice tuhosti.
  - Aktualizace šablony pro daný řádek.
  - Načtení jednoho řádku matice v původním formátu.
  - Vytvoření jednoho řádku matic $A_{11}$, $A_{22}$, $A_{33}$.
  - Uložení známých prvků pro pozdější použití do kruhových bufferů podle šablony.
  - Vytvoření jednoho řádku matic $A_{12}$, $A_{13}$, $A_{23}$. Prvky poloviny řádku každé matice jsou známy při načtení řádku matice v původním formátu a prvky druhé poloviny jsou uloženy v kruhových bufferech.
  - Uložení jednoho řádku všech matic $A_{11}$, $A_{22}$, $A_{33}$, $A_{12}$, $A_{13}$, $A_{23}$ do datových souborů.
- Konec cyklu přes všechny řádky matice tuhosti.
- Uzavření datových souborů.

*Konverze pravé strany*

- Otevření příslušných datových souborů.
- Načtení pravé strany do bufferu.
- Uložení složky posunutí `x` pravé strany do datového souboru.
- Uložení složky posunutí `y` pravé strany do datového souboru.
- Uložení složky posunutí `z` pravé strany do datového souboru.
- Uzavření datových souborů.

*Konec programu*

## 5.2 Sekvenční řešič PCG–2

Program PCG–2 slouží k řešení soustavy lineárních rovnic

$$Au = f$$

metodou sdružených gradientů s předpodmíněním, přičemž datové struktury soustavy, tj. matice tuhosti $A$, pravá strana $f$ a také hledané řešení $u$, jsou uloženy v blokovém formátu, kde mají separovány složky posunutí. Řešení soustavy je počítáno podle sekvenčního algoritmu (viz. kapitola Sdružené gradienty), program je určen k použití na jednom počítači.

Při výpočtu se pracuje se soubory, které uvádí následující seznam:

| | | |
|---|---|---|
| `FKBCXX.G32` | ... | matice $A_{11}$, |
| `FKBCXY.G32` | ... | matice $A_{12}$, |
| `FKBCXZ.G32` | ... | matice $A_{13}$, |
| `FKBCYY.G32` | ... | matice $A_{22}$, |
| `FKBCYZ.G32` | ... | matice $A_{23}$, |
| `FKBCZZ.G32` | ... | matice $A_{33}$, |
| `FRBCX.G32` | ... | pravá strana $f_x$, |
| `FRBCY.G32` | ... | pravá strana $f_y$, |
| `FRBCZ.G32` | ... | pravá strana $f_z$, |
| `FV.G32` | ... | údaje popisující úlohu, |
| `FUX.G32` | ... | hledané řešení $u_x$, |
| `FUY.G32` | ... | hledané řešení $u_y$, |
| `FUZ.G32` | ... | hledané řešení $u_z$, |
| `PCG.REP` | ... | podrobný komentář výpočtu, |
| `MES32.REP` | ... | stručný komentář výpočtu. |

Tento program je odladěn a vyzkoušen na počítačích typu PC a na pracovních stanicích IBM RS/6000. Na disketě se v adresáři `PCG-2` nacházejí tyto soubory:

◇ `ITERA.FOR`,

zdrojový text hlavního programu, který vytváří prostředí pro spuštění řešiče, nejprve jsou načteny údaje popisující úlohu voláním funkce `READFV` a ostatní doplňující údaje potřebné pro výpočet uživatel zadá z klávesnice, potom se všechny údaje předají rutině `PCG`, která provede výpočet řešení soustavy,

◇ `PCG.FOR`,

zdrojový text rutiny, která řeší metodou sdružených gradientů soustavu lineárních rovnic $Au = f$ podle sekvenčního algoritmu, z této rutiny jsou volány rutiny `MXV`, `PCINIT`, `PCSOLV`, `PARSOL` a `RTIME`,

◇ `MXV.FOR`,

zdrojový text operace násobení matice krát vektor $w \leftarrow A\ v$, která se provádí v každé iteraci, zároveň počítá skalární součin $s$ vektorů $v$ a $w$, $s =< v, w >=< v, A\ v\ >$,

◇ `PCINIT.FOR`,

zdrojový text rutiny, která slouží k inicializaci předpodmiňovače,

◇ `PCSOLV.FOR`,

zdrojový text předpodmínění, tato rutina v každé iteraci řeší soustavu $Cw = re$, resp. $w = C^{-1}re$,

◇ `PARSOL.FOR`,

zdrojový text počáteční aproximace „částečná řešení“, pokud je zvolena počáteční aproximace „částečná řešení“, rutina `PCG` v přípravné fázi algoritmu volá rutinu `PARSOL`, která vypočte počáteční aproximaci $u_{init}$, z této rutiny se volají podprogramy `INITPS` a `MXVPS`,

◇ `INITPS.FOR`,

zdrojový text rutiny, která slouží pro inicializaci počáteční aproximace „částečná řešení“,

◇ `MXVPS.FOR`,

zdrojový text operace násobení matice krát vektor při výpočtu počáteční aproximace $u_{init}$ metodou „částečných řešení“,

◇ `READFV.FOR`,

zdrojový text rutiny pro načtení údajů ze souboru `FV.G32`, které popisují úlohu, tato rutina je univerzální pro všechny programy,

◇ `RTIME.FOR`,

zdrojový text funkce, která přečte systémový čas, tato funkce je volána na začátku a na konci úseku programu, jehož dobu trvání chceme změřit, rozdíl mezi hodnotami, které funkce vrátí pro jednotlivá její volání, je celkový čas měřeného úseku programu v sekundách (čas „nástěnných hodin“),

◇ `PCG-2.BAT`,

dávkový soubor pro překlad programu na platformě PC,

◇ `PCG-2.LNK`,

pomocný soubor pro překlad programu na platformě PC,

◇ `PCG-2.WS`,

dávkový soubor pro překlad programu na platformě WS.

## 5.3 Paralelní řešič PCG–3

Program PCG–3 slouží k řešení soustavy lineárních rovnic

$$Au = f$$

metodou sdružených gradientů s předpodmíněním, přičemž datové struktury soustavy, tj. matice tuhosti $A$, pravá strana $f$ a také hledané řešení $u$, jsou uloženy v blokovém formátu, kde mají separovány složky posunutí. Řešení soustavy je počítáno podle paralelního algoritmu (viz. kapitola Paralelizace sdružených gradientů), program je určen k použití na několika počítačích současně, které při výpočtu řešení spolupracují.

Program pracuje se stejnými soubory jako jeho sekvenční verze, program PCG–2. Výhodou paralelního programu je rozložení paměťové náročnosti mezi jednotlivé počítače ve virtuálním stroji. To se projevuje dvěma způsoby. Vektory $u$, $v$, $w$, $r$, $re$, se kterými algoritmus pracuje, mají v každém z uzlů třetinovou velikost oproti velikosti stejných vektorů v sekvenčním programu. Druhou výhodou je, že datové soubory úlohy nemusí být všechny na každém počítači, na kterém běží nějaký z uzlů, ale mohou být rozděleny mezi počítače podle toho, který uzel na nich běží, tímto způsobem:

| *Uzel X:* | *Uzel Y:* | *Uzel Z:* |
|---|---|---|
| FKBCXX.G32 | FKBCXY.G32 | FKBCXZ.G32 |
| FKBCXY.G32 | FKBCYY.G32 | FKBCYZ.G32 |
| FKBCXZ.G32 | FKBCYZ.G32 | FKBCZZ.G32 |
| FRBCX.G32 | FRBCY.G32 | FRBCZ.G32 |
| FUX.G32 | FUY.G32 | FUZ.G32 |

Tento program je odladěn a vyzkoušen na pracovních stanicích IBM RS/6000, které jsou zapojeny do virtuálního stroje pracujícího pod prostředím PVM. Na disketě se v adresáři PCG-3 nacházejí tyto soubory:

◇ ITERA.FOR,

zdrojový text hlavního programu, který vytváří prostředí pro řídící rutinu řešiče, nejprve spustí všechny uzly, potom volá rutinu READFV, která načte údaje popisující úlohu a další informace ovlivňující běh programu jsou uživatelem zadány z klávesnice, potom se řízení a všechny údaje předají rutině PCG,

◇ PCG.FOR,

zdrojový text řídící rutiny řešiče, odtud jsou volány rutiny MXV, PCINIT a RTIME,

◇ MXV.FOR,

zdrojový text operace násobení matice krát vektor na straně řídící úlohy, rutina slouží pro synchronizaci uzlů,

◇ PCINIT.FOR,

zdrojový text inicializace předpodmiňovače na straně řídící úlohy, která slouží k synchronizaci uzlů,

◇ READFV.FOR,

zdrojový text rutiny pro načtení údajů ze souboru FV.G32, které popisují úlohu, tato rutina je univerzální pro všechny programy,

◇ `RTIME.FOR`,

zdrojový text funkce, která přečte systémový čas, tato funkce je volána na začátku a na konci úseku programu, jehož dobu trvání chceme změřit, rozdíl mezi hodnotami, které funkce vrátí pro jednotlivá její volání, je celkový čas měřeného úseku programu v sekundách (čas „nástěnných hodin“),

◇ `ITERAN.FOR`,

zdrojový text hlavního programu, který vytváří prostředí pro uzel, po navázání spojení uzlu s řídící úlohou se řízení předá do podprogramu `PCGN`,

◇ `PCGN.FOR`,

zdrojový text řešiče na straně uzlu, odtud jsou volány volány rutiny `MXVN`, `PCINITN`, `PCSOLV` a `PARSOL`,

◇ `MXVN.FOR`,

zdrojový text operace násobení matice krát vektor $w_i \leftarrow A\, v_i$ na straně uzlu, rutina před komunikací s řídící úlohou volá podprogramy `SMDXV` a dvakrát `SMNXV`,

◇ `SMDXV.FOR`,

zdrojový text operace násobení diagonální submatice krát vektor na straně uzlu,

◇ `SMNXV.FOR`,

zdrojový text operace násobení nediagonální submatice krát vektor na straně uzlu,

◇ `PCINITN.FOR`,

zdrojový text inicalizace předpodmiňovače na straně uzlu,

◇ `PCSOLV.FOR`,

zdrojový text předpodmínění na straně uzlu, rutina v každé iteraci řeší soustavu $C_i w_i = re_i$, resp. $w_i = C_i^{-1} re_i$,

◇ `PARSOL.FOR`,

zdrojový text počáteční aproximace $u_{init,i}$ „částečná řešení“ na straně uzlu, rutina volá podprogramy `INITPS` a `SMDXV`,

◇ `INITPS.FOR`,

zdrojový text inicializace počáteční aproximace „částečná řešení“ na straně uzlu,

◇ `PCG-3.WS`,

dávkový soubor pro překlad na platformě WS (+ PVM),

◇ `PVMHOST.CFG`,

příklad konfiguračního souboru pro prostředí PVM.

Pro úplnost je na další straně uvedena v tabulce 1 veškerá komunikace, která probíhá mezi řídící úlohou a uzly. Jednotlivé sloupce tabulky zleva postupně obsahují třídu zprávy, typ komunikace (**ř**ídící/**d**atová), krátký popis zprávy, informaci, mezi kterými rutinami a kterým směrem je zpráva posílána a strukturu zprávy.

Každá položka ve struktuře zprávy se skládá ze tří částí. Nejprve je uvedeno, o jaký typ dat se jedná a jejich dálka v bajtech, např. `I2` jsou celočíselná (integer), dvoubajtová data, `R8` jsou reálná (real), osmibajtová čísla. Ve druhé části je jméno proměnné, tedy adresa, odkud se budou data brát nebo kam se budou ukládat, a v poslední části je uveden počet dat.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | d | pole `taskids` | ITERA | → | ITERAN | I4 | numt | 1 |
| | | | | | | I4 | tids | 3 |
| 2 | d | údaje popisující úlohu | ITERA | → | ITERAN | I4 | iap | 1 |
| | | | | | | I4 | ipc | 1 |
| | | | | | | I4 | nd | 1 |
| | | | | | | I4 | nn | 1 |
| | | | | | | I2 | nx | 1 |
| | | | | | | I2 | ny | 1 |
| | | | | | | I2 | nz | 1 |
| 5 | ř | řízení výpočtu uzlů | PCG | → | PCGN | I4 | pt | 1 |
| 6 | ř | potvr. ukonč. predpod. | PCGN | → | PCG | I4 | pt | 1 |
| 7 | ř | potvrzení ukončení | PCGN | → | PCG | I4 | iterap | 1 |
| | | počáteční aproximace | | | | R4 | tolap | 1 |
| | | (pouze pro IAP=2) | | | | | | |
| 50 | d | koeficient $a$ | PCG | → | PCGN | R4 | a | 1 |
| 51 | d | koeficient $b$ | PCG | → | PCGN | R4 | b | 1 |
| 52 | | | | | | | | |
| 55 | d | vektory příspěvků | MXV | → | MXVN | R4 | buf(x) | 2∗NN |
| | | při násobení matice | | | | R4 | buf(y) | 2∗NN |
| | | krát vektor | | | | R4 | buf(z) | 2∗NN |
| 59 | d | typ předpodmínění | PCINIT | → | PCINITN | I4 | ipc | 1 |
| | | (pouze pro IPC=3,11) | | | | | | |
| 60 | d | skalární součiny | PCGN | → | PCG | R8 | sffi | 1 |
| | | | | | | R8 | si | 1 |
| | | | | | | R8 | srr0i | 1 |
| | | | | | | R8 | srw0i | 1 |
| 61 | d | skalární součin | PCGN | → | PCG | R8 | si | 1 |
| 62 | d | skalární součiny | PCGN | → | PCG | R8 | srri | 1 |
| | | | | | | R8 | srwi | 1 |
| 65 | d | vektory příspěvků | MXVN | → | MXV | I4 | k | 1 |
| | | při násobení matice | | | | I4 | l | 1 |
| | | krát vektor | | | | R4 | buf(k) | NN |
| | | | | | | R4 | buf(l) | NN |
| 69 | d | výstup inicializace | PCINITN | → | PCINIT | I4 | ipci | 1 |
| | | předpodmiňovače | | | | I4 | nzeri | 1 |
| | | (pouze pro IPC=3,11) | | | | | | |

Tabulka 1: Komunikace mezi řídící úlohou a uzly

# 6 Testování

Pro testování programů je důležitá znalost testovacích úloh, na základě níž se dají stanovit předběžné odhady výsledků nebo chování programů. Stejně důležité pro celkové zhodnocení testů je uvádět konfigurace počítačů a nastavení přepínačů, které běh programů ovlivňují.

Konfigurace počítačů, které byly při testech použity, jsou uvedeny v kapitole Systémové prostředky. Řešiče PCG–2 a PCG–3 mají shodné možnosti ovlivnění výpočtu řešení. V obou programech jsou čtyři přepínače, kterými lze měnit běh programů a které uživatel zadává po spuštění programu z klávesnice.

Prvním z přepínačů je relativní přesnost výpočtu `EPS`, která se používá v algoritmu sdružených gradientů ve fázi testování pro ukončení běhu programu. Uvádí se ve formě reálného čísla, běžně se používají hodnoty $10^{-3}$, $10^{-5}$.

Dále uživatel zadává typ předpodmínění `IPC`. Může být uvedena jedna z možností: 0 pro výpočet bez předpodmínění, 1 pro diagonální předpodmínění, 3 pro předpodmínění DD-IF a 11 pro automatický chod. Automatický chod nejprve ve fázi inicializace předpodmiňovače zkouší použít předpodmínění DD-IF. Pokud je DD-IF předpodmínění nestabilní, program použije diagonální předpodmínění.

Třetí volbou je maximální počet iterací `IMAX`. Pokud program při výpočtu řešení provede `IMAX` iterací, bude běh programu ukončen i přesto, že hledané řešení dosud nebylo nalezeno. Do datových souborů se pak ukládají aktuální hodnoty vektoru neznámých z poslední iterace.

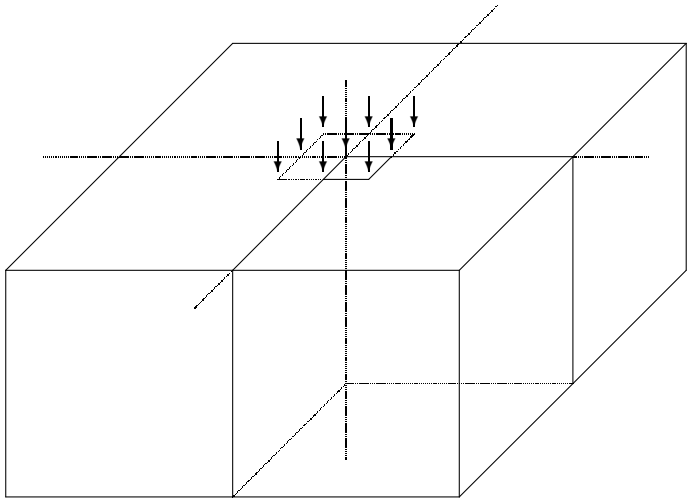
Posledním přepínačem je typ počáteční aproximace `IAP`, který může být jedna z možností: 0 pro nulovou počáteční aproximaci pro výpočet bez předpodmínění nebo pro počáteční aproximaci odvozenou z pravé strany pro výpočet s předpodmíněním, 1 pro případ načtení počáteční aproximace z datových souborů a 2 pro počáteční aproximaci, která je produktem „částečných řešení“.

## 6.1 Testovací úlohy

Pro testování řešičů byly použity modelové úlohy z geomechaniky. V testech začínají názvy úloh navíc písmenem `B`, které zdůrazňuje, že datové struktury úlohy byly převedeny programem DSPLIT do blokového formátu.

### 6.1.1 Úloha STRIPE

V úloze STRIPE je popsána deformace podloží základu, který je modelován silovým zatížením, viz. obr. 14. Uvažovaná oblast je kvádr o velikosti 100 x 100 x 40 metrů, která je na horní straně uprostřed stejnoměrně zatížena ve čtvercové oblasti o velikosti 10 x 10 metrů. Zátěž představuje tlak 2,4 MPa.



Obr. 14: Zatížení podloží základu

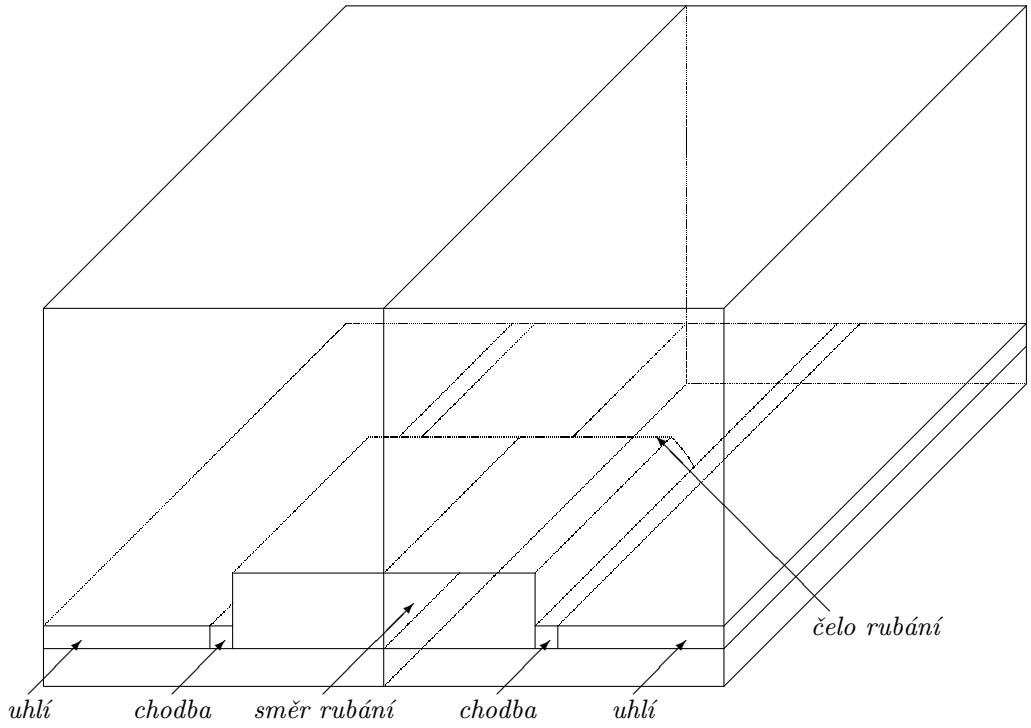
Z obrázku je patrná symetrie oblasti a zatížení, proto můžeme uvažovat pouze čtvrtinu původní oblasti o rozměrech 50 x 50 x 40 metrů, v ostatních částech bude řešení stejné. Oblast bude zatížená v rohu horní strany na čtvercové ploše 5 x 5 metrů. Vnitřní síly jsou určeny váhou materiálu. Hraniční podmínky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

Soustava lineárních rovnic vznikla diskretizací oblasti pravidelnou sítí (viz. obr. 1 vpravo) a použitím metody konečných prvků. Hustota sítě byla jemnější směrem k namáhané podoblasti. Síť měla 40 x 40 x 40 = 64000

uzlů, což představuje 192000 rovnic a paměťovou náročnost uložení datových struktur soustavy 33 MB kapacity disku.

### 6.1.2 Úloha STOPE

Úloha STOPE modeluje situaci při těžbě uhlí v uhelné sloji (viz. obr. 15).




Obr. 15: Dobývání uhlí

Oblast je ve tvaru kvádru o velikosti 300 x 150 x 150 metrů a je umístěna ve hloubce 500 metrů pod povrchem. Hloubka je na modelu simulovaná zatížením 12,5 MPa na celou horní stranu oblasti. V uhelné sloji jsou vyraženy dvě rovnoběžné chodby, mezi kterými probíhá těžba uhlí. V určité vzdálenosti za porubní frontou (v modelu nulová vzdálenost) porušený materiál zasypává vytěženou oblast.

Z obrázku je patrná symetrie oblasti, proto můžeme uvažovat pouze polovinu původní oblasti o velikosti 150 x 150 x 150 metrů, ve druhé polovině oblasti bude řešení stejné. Vnitřní síly jsou určeny váhou materiálu. Hraniční podmínky reprezentují nulová normálová posunutí a nulové tangenciální tlaky. V oblasti jsou tři různé materiály: uhlí, hornina, která uhlí obklopuje, a porušený materiál po těžbě. Chodby jsou modelovány nulovým materiálem. Všechny materiály jsou izotropní.

Soustava lineárních rovnic vznikla diskretizací oblasti pravidelnou sítí (viz. obr. 1 vpravo) a použitím metody konečných prvků. Hustota sítě byla jemnější v oblasti těžby. Síť měla stejný počet uzlů jako v úloze STRIPE, 40 x 40 x 40 = 64000. Proto je stejný i rozsah úlohy, celkový počet rovnic 192000 a paměťová náročnost uložení soustavy 33MB diskové kapacity.

### 6.1.3 Ostatní úlohy

Ostatní úlohy, které byly použity v testech, jsou stejného typu jako úloha STRIPE. Uvažovaná oblast ve tvaru kvádru je na horní straně v rohu zatížena. Soustava vznikla diskretizací úlohy pravidelnou sítí o 5 x 5 x 5 = 125 uzlech. Celkový počet rovnic je 375, které uložené na disku zabírají méně než 100 kB.

Typ diskretizační sítě pro úlohy CPA a SYS je zachycen na obr. 1 vlevo a pro úlohu CPAS na stejném obrázku vpravo. Úloha CPB se od ostatních odlišuje tím, že existuje její přesné řešení.

## 6.2 Počáteční aproximace

První testy byly zaměřené na posouzení efektivity nové algoritmické varianty sdružených gradientů, ve které se počáteční aproximace $u_{init}$ počítá metodou „částečných řešení“.

„Částečná řešení“ počítají metodou sdružených gradientů s diagonálním předpodmíněním jednotlivé složky posunutí $u_{init,i}$ pro $i = x, y, z$. Změnou relativní přesnosti výpočtu $\varepsilon$ metody „částečných řešení“ lze ovlivnit počáteční aproximaci $u_{init}$ a tím i výpočet řešení celé soustavy lineárních rovnic.

V tabulce 2 jsou výsledky vlivu relativní přesnosti výpočtu $\varepsilon$ „částečných řešení“ na celkovou dobu řešení soustavy lineárních rovnic velmi malého rozsahu.

Testování bylo provedeno na počítači typu PC sekvenčním řešičem (programem PCG–2) pro relativní přesnost řešení soustavy `EPS=0.00001`, maximální počet iterací `IMAX=1000`. Výpočet probíhal bez předpodmínění, `IPC=0`. První řádek tabulky je výsledek výpočtu řešení soustavy pro počáteční aproximaci odvozenou z pravé strany soustavy, `IAP=0`. Ostatní řádky jsou výsledky výpočtu řešení soustavy s typem počáteční aproximace „částečná řešení“, `IAP=2`, při různé relativní přesnosti výpočtu $\varepsilon$ „částečných řešení“. Jednotlivé sloupce tabulky znamenají postupně zleva: $\varepsilon$ relativní přesnost výpočtu „částečných řešení“, číslo iterace, ve které bylo nalezeno řešení soustavy při zadané přesnosti `EPS`, čas výpočtu počáteční aproximace a celkový čas potřebný pro výpočet řešení soustavy.

| $\varepsilon$ | **BCPA** | | | **BCPAS** | | |
|---|---|---|---|---|---|---|
| | It. | Čas apr. [sec.] | Čas celk. [sec.] | It. | Čas apr. [sec.] | Čas celk. [sec.] |
| - | 66 | - | 22,133 | 85 | - | 28,340 |
| $10^{0}$ | 43 | 0,379 | 14,879 | 56 | 0,383 | 19,000 |
| $10^{-1}$ | 33 | 0,547 | 11,867 | 48 | 0,660 | 16,750 |
| $10^{-2}$ | 30 | 0,660 | 10,992 | 39 | 0,988 | 14,230 |
| $10^{-3}$ | 20 | 0,879 | 8,020 | 19 | 1,148 | 7,961 |
| $10^{-4}$ | 3 | 1,152 | 2,863 | 11 | 1,320 | 5,609 |
| $10^{-5}$ | 0 | 1,211 | 1,980 | 1 | 1,543 | 2,641 |

| $\varepsilon$ | **BCPB** | | | **BSYS** | | |
|---|---|---|---|---|---|---|
| | It. | Čas apr. [sec.] | Čas celk. [sec.] | It. | Čas apr. [sec.] | Čas celk. [sec.] |
| - | 12 | - | 4,832 | 43 | - | 14,770 |
| $10^{0}$ | 11 | 0,441 | 4,723 | 42 | 0,332 | 14,500 |
| $10^{-1}$ | 11 | 0,441 | 4,723 | 33 | 0,551 | 11,922 |
| $10^{-2}$ | 12 | 0,551 | 5,168 | 29 | 0,711 | 10,762 |
| $10^{-3}$ | 12 | 0,820 | 5,430 | 20 | 0,883 | 8,082 |
| $10^{-4}$ | 12 | 0,930 | 5,539 | 3 | 1,102 | 2,859 |
| $10^{-5}$ | 12 | 0,988 | 5,539 | 0 | 1,211 | 1,980 |

Tabulka 2: Testování počáteční aproximace na malých úlohách

Z tabulky je patrné výrazné zlepšení konvergence metody sdružených gradientů při použití „částečných řešení“ u všech úloh kromě úlohy CPB, pro kterou existuje přesné řešení. U ostatních úloh při zmenšování hodnoty $\varepsilon$ došlo k podstatnému snížení počtu iterací a tedy i celkového času potřebného pro nalezení řešení soustavy.

Při řešení větších úloh jsou podstatné další vlivy. Patří mezi ně dostatečná velikost operační paměti počítače, rychlost komunikace mezi procesorem a vnější pamětí, a další. Proto výsledky mohou být značně rozdílné, jak ukáže následující tabulka 3.

| $\varepsilon$ | **BSTRIPE** | | | **BSTOPE** | | |
|---|---|---|---|---|---|---|
| | It. | Čas apr. [sec.] | Čas celk. [sec.] | It. | Čas apr. [sec.] | Čas celk. [sec.] |
| - | 49 | - | 1093,550 | 63 | - | 1561,000 |
| $10^{-1}$ | 42 | 162,120 | 1089,670 | 62 | 160,750 | 1503,130 |
| $10^{-2}$ | 42 | 257,620 | 1184,990 | 62 | 271,030 | 1792,720 |
| $10^{-3}$ | 42 | 330,550 | 1258,140 | 62 | 332,650 | 1853,150 |

Tabulka 3: Testování počáteční aproximace na větších úlohách

Testování proběhlo na pracovní stanici IBM RS/6000 `fany`, úlohy byly řešeny programem PCG–2 pro relativní přesnost řešení soustavy `EPS=0.001`, maximální počet iterací `IMAX=1000` a s předpodmíněním DD-IF, `IPC=3`. Tabulka je uspořádána stejným způsobem jako tabulka 2.

Z tabulky vyplývá, že u větších úloh sice může počáteční aproximace „částečná řešení“ jisté zlepšení přinést, ale toto zlepšení není příliš podstatné. Navíc se ukazuje, že při přiblížení hodnoty relativní přesnosti $\varepsilon$ k nule, je nárůst času při výpočtu počáteční aproximace příliš velký, proto je použití „částečných řešení“ neefektivní. Místo poklesu se projevuje nárůst celkové doby výpočtu řešení soustavy. Z toho lze usoudit, že se nevyplatí snižovat hodnotu $\varepsilon$. Její optimální velikost je přibližně `0.1`. „Částečná řešení“ lze vylepšit změnou předpodmínění, které je při výpočtu počáteční aproximace metodou sdružených gradientů použito. Místo diagonálního předpodmínění je možné použít předpodmínění DD-IF, čímž se dosáhne lepší konvergence sdružených gradientů, zmenšení doby potřebné pro výpočet počáteční aproximace a tím i zmenšení celkového času výpočtu řešení soustavy. Přes toto

možné zlepšení nebude počáteční aproximace „částečná řešení“ příliš efektivní, neboť odhadovaná úspora času budu méně než 10 % .

## 6.3 Efektivita paralelizace

K obvyklým charakteristikám, které se používají k posouzení efektivity paralelního programu, patří zrychlení $S_0$ (anglicky speed up) a účinnost $E$ (anglicky efficiency). Zrychlení se vypočítá podle vztahu

$$S_0 = \frac{T_0}{T_N} \leq N,$$

kde $T_0$ je čas „nejlepšího“ sekvenčního programu, který provádí stejný výpočet se stejnými daty jako paralelní program a $T_N$ je čas paralelního programu běžícího na $N$ procesorech. Maximální hodnota zrychlení paralelního programu je rovna počtu procesorů, na kterých byl program spuštěn. Ze zrychlení je možné spočítat účinnost

$$E = \frac{S_0}{N},$$

přičemž implementace paralelního programu je tím lepší, čím více se účinnost blíží hodnotě jedna. Do úvahy lze vzít i zrychlení

$$S_1 = \frac{T_1}{T_N},$$

kde $T_1$ je čas výpočtu paralelního programu, který běží na jednom procesoru.

Pro testování efektivity paralelního programu PCG–3 pro řešení soustavy lineárních rovnic byl používán klastr tří pracovních stanic IBM RS/6000 `prokop`, `geam` a `fany` (viz. kapitola Systémové prostředky). Jednotlivé počítače jsou výkonově značně rozdílné, uvedené výsledky jsou vztaženy k počítači `geam`, jehož výkon je nejbližší průměru výkonů všech počítačů.

Přepínače ovlivňující běh programu byly nastaveny ve všech testech takto: relativní přesnost výpočtu řešení soustavy `EPS=0.00001`, maximální počet iterací `IMAX=1000`, typ předpodmínění DD-IF `IPC=3` a počáteční aproximace odvozená z pravé strany `IAP=0`.

V tabulce 4 jsou výsledky jednotlivých testů, které byly provedeny na větší úloze. Význam jednotlivých sloupců tabulky je postupně zleva následující: označení situace, při které byl test proveden, počet procesorů, na kterých

byl program spuštěn, číslo iterace, při které bylo nalezeno řešení soustavy při dané relativní přesnosti výpočtu `EPS`, celkový čas výpočtu řešení soustavy, zrychlení $S_0$, účinnost $E$ a zrychlení $S_1$.

| **BSTOPE** | | | | | | |
|---|---|---|---|---|---|---|
| Sit. | Procesorů $N$ | It. | Celk. čas [sec.] | Zrychlení $S_0$ | Účinnost $E$ | Zrychlení $S_1$ |
| A | 1 | 98 | 13514 | - | - | - |
| B | 1 | 98 | 26951 | 0,50 | - | - |
| C | 2 | 98 | 17976 | 0,75 | 0,38 | 1,50 |
| D | 2 | 98 | 16427 | 0,82 | 0,41 | 1,64 |
| E | 3 | 98 | 6950 | 1,94 | 0,65 | 3,88 |

Tabulka 4: Testování efektivity paralelního řešiče

V testech paralelního programu byla řídící úloha paralelního programu spuštěna vždy na počítači `geam`. Jednotlivé situace, při kterých byly testy provedeny, jsou:

- Písmeno A označuje situaci, kdy výpočet byl proveden sekvenčním programem PCG–2, který běžel na počítači `geam`. Vůči výsledkům tohoto testu jsou vztaženy výsledky všech ostatních testů, které byly provedeny paralelním programem. Celkový čas výpočtu řešení za těchto podmínek budeme považovat za čas „nejlepšího“ sekvenčního programu $T_0$.

- Písmeno B znamená, že všechny tři uzly paralelního programu, které počítají jednotlivé složky posunutí, běží na počítači `geam`. V této situaci běží paralelní program na jednom procesoru, proto celkový čas výpočtu řešení soustavy je $T_1$.

- Písmenem C je označena situace, kdy dva z uzlů běží na počítači `geam` a jeden uzel běží na počítači `prokop`. Tento test byl uskutečněn pro vyzkoušení, zda i výrazně pomalejší počítač v klastru může pomoci zrychlit běh paralelního programu.

- Situace označená písmenem D znamená, že dva z uzlů běží na počítači `geam` a třetí uzel běží na počítači `fany`. Výsledky tohoto testu slouží k porovnání s výsledky testu označeného písmenem C.

- Poslední situace označená písmenem E znamená, že na počítači `geam` běží pouze jeden uzel a zbylé dva běží na počítači `fany`. Protože počítač `fany` je výrazně výkonnější než počítač `geam`, tato situace zároveň simuluje běh paralelního programu na multiprocesorovém stroji s distribuovanou pamětí (například IBM SP–2), kde jsou všechny počítače stejně výkonné. Takové stroje však navíc obsahují prostředky pro rychlou komunikaci úloh běžících na jednotlivých procesorech, které vyplývají z architektury počítače. Proto by byl výsledný celkový čas řešení soustavy menší. Přes uvedené odlišnosti budeme tuto situaci při výpočtech zrychlení a účinnosti považovat za stejnou, jako kdyby každý uzel běžel na jednom procesoru multiprocesorového počítače, kde všechny procesory mají stejný výkon, počet použitých procesorů tedy bude $N = 3$ (ve skutečnosti $N = 2$).

Z tabulky výsledků dosažených při testování je patrné, že realizovaný paralelní program je efektivní při použití tří procesorů (situace E s uvedenými připomínkami). V tomto případě je zrychlení paralelního programu oproti jeho sekvenční verzi téměř dvojnásobné, $S_0 = 1,94 \doteq 2$, což je při řešení rozsáhlých soustav lineárních rovnic velmi příznivé. Přitom paměťová náročnost problému je na klastru pracovních stanic rovnoměrně rozložena. Účinnost realizované paralelní implementace sdružených gradientů je $E = 0,65$. Tato hodnota není příliš vysoká, vyplývá z architektury klastru. Přístup procesoru k zpracovávaným datům, která nemá ve své operační paměti, je velmi pomalý, v našem případě je realizován prostřednictvím lokální sítě. Takové „křížové“ přístupy se musí provádět v každé iteraci zejména při provádění operace násobení matice krát vektor, kde dochází k přenosu většího objemu dat. Tím dochází ke značnému zpomalení běhu programu a klesá účinnost. Zrychlení programu při použití tří procesorů oproti běhu programu na jednom procesoru je $S_1 = 3,88 \doteq 4$, tedy téměř čtyřnásobné.

Z hodnot zrychlení a účinnosti vyplývá, že se nevyplatí paralelní program spouštět na menším počtu procesorů než tři. Při použití dvou procesorů je celkový čas výpočtu paralelního programu vyšší než u sekvenčního programu. Významnější je poznatek, že k řešení problému, které je realizováno paralelním programem na klastru pracovních stanic, může svým výkonem výrazně

přispět i relativně pomalejší počítač. Z porovnání celkových časů řešení v situacích C a D je vidět, že oba časy jsou srovnatelné. Liší se pouze proto, že výkon počítače `prokop` je příliš nízký oproti výkonu počítače `geam`, přesto rozdíl v celkové době výpočtu je relativně malý.

# 7 Závěr

Cílem diplomové práce byla paralelní implementace metody sdružených gradientů s využitím separace složek posunutí. Proto byl nejprve vytvořen konverzní program pro uložení datových struktur soustavy lineárních rovnic s oddělenými složkami posunutí.

Dále byla programově realizována sekvenční verze sdružených gradientů, která s takto uloženými datovými strukturami pracuje. V tomto programu byla prakticky vyzkoušena nová algoritmická varianta počáteční aproximace nazvaná „částečná řešení“. Tato počáteční aproximace může z hlediska rychlosti konvergence sdružených gradientů přinést zlepšení, které však není příliš významné.

Posledním vytvořeným programem byla paralelní implementace sdružených gradientů, při jejíž realizaci a testování byl použit klastr pracovních stanic IBM RS/6000 pracující pod prostředím PVM. Prostředí PVM se ukázalo jako silný prostředek pro spřažení (i heterogenních) pracovních stanic, na kterém lze používat paralelní programy vytvořené podle modelu posílání zpráv.

Oba řešiče byly prakticky vyzkoušeny na modelových úlohách z geomechaniky. Při testech se projevila efektivita paralelní verze řešiče, neboť paralelní verze řešiče byla při stejném výpočtu téměř dvakrát rychlejší než verze sekvenční. I přes nízkou hodnotu účinnosti paralelní implementace sdružených gradientů, která vyplývá především z architektury použitého výpočetního prostředku, je vidět, proč zájem o spřažené pracovní stanice roste. Důležité je také zjištění, že při výpočtu prováděném na takovém výpočetním systému může pomoci i výkonově slabší počítač.

Přes příznivé výsledky, které v testech dosahoval paralelní řešič, existují další možnosti jeho zlepšení. Při použití stejných systémových prostředků je to především optimalizace rutiny násobení matice krát vektor zejména z hlediska komunikace.

# Literatura

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

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

[3] R. Blaheta: *Displacement decomposition – incomplete factorization preconditioning techniques for linear elasticity problems,* Num. Lin. Alg. Appl. 1 (1994), 107-126.

[4] R. Blaheta, R. Kohut: *PCG–1 iterative solver,* Rep. 9501, Dept. Appl. Math., Institute of Geonics AS CR, Ostrava 1995.

[5] R. Blaheta: *Additive and multiplicative preconditioning for elasticity problems,* konf. VŠB, Ostrava 1995.

[6] V. Lednický: *Efektivní implementace sdružených gradientů s předpodmíněním,* diplomová práce, katedra aplikované matematiky, VŠB–TU Ostrava.

[7] Al Geist et al.: *PVM 3 user's guide and reference manual,* Oak Ridge Nat. Lab., Oak Ridge 1993.

# Seznam obrázků



# Seznam tabulek

# Obsah