Západočeská univerzita Plzeň

Fakulta aplikovaných věd

Samostatná práce

z

PPR zadání č. 3

Vypracoval: **Petr Muchna (A06095)**
27.6. 1984
pmuchna@gmail.com

Zadání

Realizujte výpočet **k**-té mocniny celočíselné matice (n,n).

Zadanou úlohu individuálně vypracujte ve dvou verzích:

1. Paralelní program pro systém se sdílenou pamětí - jazyk Java
2. Paralelní program pro systém s distribuovanou pamětí MPI

Popis algoritmu

Úlohu jsem řešil paralelizací jednoho násobení. Elementární částí práce je prvek na určité pozici v matici součinu. Hodnota tohoto prvku se stanoví jako skalární součin příslušného řádku z násobence a sloupce z násobitele. Násobitelem bude vždy vstupní matice, násobenec se bude měnit s výpočtem určitého stupně mocniny.

Systém se sdílenou pamětí - Java

Popis řešení

Úloha byla realizována dle modelu farmer – worker. Pro paralelizaci výpočtu se používá pouze skalární součin, nikoliv celé násobení dvou matic. Farmer tedy musí zajistit řízení umocňovaní, worker pouze počítá skalární součiny řádků a sloupců matice. Farmer je reprezentován třídou `Monitor`, která poskytuje metody pro rozdělování práce a zpracování výsledků od workerů (třída `Worker`). Inicializace probíhá tak, že jsou spuštěny jednotlivé thready workerů, kterým je předána reference na zdrojovou matici násobitele a matici mezivýsledku násobence. Zdrojovou matici lze při spuštění zadat ze souboru, z konzole nebo případně vygenerovat náhodně. To je vhodné zejména pro testování efektu paralelizace a „automatic load balancingu“, lze zadat matici o velké dimenzi a sledovat čas zpracování úlohy. Po spuštění požádá worker o práci (*metoda* `getNextWorkPart`). Objem práce je reprezentován třídou WorkPart, pomocí této třídy se do workeru předá souřadnice počátečního prvku a počet prvků, které má worker zpracovat. Farmer si prostřednictvím celočíselné proměnné udržuje informaci o tom, který prvek bude následovat pro zpracování. Po výpočtu předá Worker výsledky farmeru (metoda `updateResult`) a požádá o další. Veškeré metody, které volá worker z Farmeru jsou typu synchronized, může do nich tedy vstoupit pouze jeden worker.

Po ukončení výpočtu násobení musí farmer dodat do workeru novou matici mezivýsledku pro zajištění dalšího stupně umocňování. Ukončení násobení indikuje worker po příjetí null z při volání metody `getNextWorkPart`. Worker poté volá metodu `getNextMupltiplicant` pro získání dalšího násobence. Některé z workerů mohou v této době ještě zpracovávat skalární součin a násobenec ve farmeru určený pro další krok mocnění nemusí být kompletní. Proto je v metodě `getNextMultiplicant` realizována bariéra. Ta zajistí, že se všechny workeři uspí, dokud nezavolá tuto metodu poslední z nich. Teprve pak je jim přidělen další násobenec a pokračuje se ve výpočtu. Farmer zajistí ukončení výpočtu po provedení posledního stupně mocniny.

Uživatelská dokumentace

Program umožňuje zadat vstupní matici třemi způsoby:

1. **Vstup ze souboru**

   Formát souboru je textový. Na první řádce se očekává dimenze matice, ostatní řádky pak obsahují jednotlivé prvky matice oddělené mezerou.

   Parametry :

   `-f <název souboru> <exponent> <počet vláken> <objem práce>`

2. **Vstup z konzole**

   Parametry :

   `-i <exponent> <počet vláken> <objem práce>`

3. **Náhodná matice**

   Tato volba je zde pro ukázku efektu „automatic load balancing“, protože při malých objemech dat se paralelizace neprojeví. V tomto případě nejde ani tak o výsledek operace, ale o dobu trvání výpočtu.

   Parametry :

   `-r <dimenze matice> <exponent> <počet vláken> <objem práce>`

Systém s distribuovanou pamětí – MPI

Popis řešení

Podobně jako při realizaci v systému se sdílenou pamětí, je i zde použit model farmer – worker a jako jednotky práce workeru jsem opět zvolil jednodušší cestu přes paralelizaci skalárního násobení. Jelikož jsem implementoval úlohu v C, nechtělo se mi z důvodu nepřehlednosti a složité inicializace používat dvourozměrná pole pro realizaci matic. Všechny matice jsou pro zjednodušení uloženy v jednorozměrných polích. Konkrétní prvek matice se pak stanoví jednoduchým výpočtem přes její dimenzi. V případě, že je nutné pro potřeby násobení získat sloupce matice, lze použít konverzní funkci `get_cols_from_row_matrix`, která převede jednorozměrné pole organizované postupně po řádcích na pole organizované po sloupcích.

Na rozdíl od PVM je v MPI jedna metoda main a chování tedy závisí na čísle procesu. Pro farmer se uvažuje proces s ID 0, ostatní workeři mají ID větší než 0. Na začátku programu je podmínka, která po detekci ID čísla procesu větví program na část pro farmer a část pro worker. Farmer nejprve inicializuje svoji strukturu parametrů (FARMER_INFO). Způsob inicializace závisí na parametrech spuštění viz. uživatelský manuál. Vzhledem k tomu, že parametrů je poměrně značné množství a jejich předávání do funkcí by způsobilo nepřehlednost, předává se pouze tato struktura, která obsahuje parametry matice a další parametry nutné pro činnost farmeru. Část z parametrů se také předává do workerů (dimenze matice a objem práce). Pro tyto účely farmer po startu posílá zprávu typu MPI_BROADCAST, aby parametry dorazili ke všem zainteresovaným workerům. Farmer dále pošle ke každému workeru inicializační data ke zpracování. Worker má provádět skalární součin,

data se posílají ve formě jednorozměrného celočíselného pole, kde jsou za sebou umístěny příslušné řádky a sloupce. Počet za sebou umístěných dvojic řádek – sloupec závisí na objemu práce. Worker zná dimenzi matice, takže provede příslušné skalární součiny a farmeru vrátí odpovídající počet prvků. Přípravu bufferu pro odeslání z farmeru obstarává funkce `get_work_part_buffer`.

Po odeslání prvních částí práce workerům čeká farmer na odpověď s výsledkem. Jakmile dorazí odpověd od libovolného workeru uloží si příslušný výsledek a pošle tomuto workeru další část práce. Pozice následujícího prvku pro odeslání se ukládá v celočíselné proměnné. Podle hodnoty této proměnné a zadaného objemu práce na jeden worker se připraví buffer pro odeslání. V případě, že byla sestavena nová výsledná matice odpovídající příslušnému stupni mocniny, je výsledek uložen a další buffer odesílaný workerům už je sestavován s touto maticí. Výpočet pokračuje, dokud se nedosáhne zadaného exponentu.

Ukončení výpočtu se musí oznámit všem workerům. Tito stále očekávají data od farmera v blokujícím RECV. K oznámení se použije zaslání zprávy se speciálním tagem. Worker podle toho pozná, že musí skončit. Vzhledem k tomu, že objem zadané práce může být nesoudělný s celkovým počtem prvků matice a zasílaná zpráva tak nemusí mít konstantní velikost, bylo zapotřebí zajistit určení počtu prvků přijímané zprávy. K tomu se použil příkaz MPI_Probe, který umožňuje zjistit informaci o zprávě před jejím samotným přijetím pomocí MPI_Recv. Jinak nebyly využity žádné pokročilejší techniky, které MPI podporuje. Charakter úlohy to nevyžadoval.

Uživatelská dokumentace

Při spuštění úlohy v MPI se předává počet vláken, formát příkazové řádky pro spuštění :

```
mpirun -np 4 mexp <parametry>
```

Pro otestování lze spustit skript :

```
./run_example.sh
```

Formát parametrů je dle typu vstupu :

1. **Vstup ze souboru**

   Formát souboru je textový. Na první řádce se očekává dimenze matice, ostatní řádky pak obsahují jednotlivé prvky matice oddělené mezerou.

   Parametry :

   `-f <exponent> <objem práce> <název souboru>`

2. **Vstup z konzole**

   Parametry :

   `-i`

3. **Náhodná matice**

Při malých objemech dat se paralelizace neprojeví. V tomto případě nejde ani tak o výsledek operace, ale o dobu trvání výpočtu.

Parametry :

```
-r <dimenze matice> <exponent> <objem práce>
```

Závěr

Během zpracování práce jsem se seznámil s MPI. Ladění úlohy v C mi zabralo dost času, ale stejně jsem nevyužil potenciál, který MPI nabízí. Patrně kvůli charakteru úlohy. Úlohu v Javě jsem odladil celkem rychle, ale nevidím na paralelizaci této úlohy příliš velké výhody. Rychlost výpočtu se nijak rapidně nezvýšila.