Párhuzamos Gauss - Seidel algoritmus ritka power system mátrixokhoz (elemzés)

Arany Kamill

2007.03.06.

Gauss - Seidel algoritmus:

$A \cdot x = b$ lineáris egyenletrendszer megoldasara használható iterációs módszer. A $n \times n$ -es ritka mátrix, x és b pedig n hosszú vektorok. $x = Cx + c$ fixpont egyenlet alakra formaljuk és kiindulva egy tetszõlges x_0 kezdõvektorból, képezzük az $x_{k+1} = Cx_k + c$ küzelítõ megoldások sorozatát. Ha x_k sorozat konvergens és tart az x^* -hoz akkor az eredeti egyenletnek is megoldása lesz. További informaciok talalhatok http://numanal.inf.elte.hu/ soveg/ oldalon az oktatási anyagok között letölthetõ jegyzetben.

Megváltoztatni egy ritka mátrixban a rendezettséget az egy permutációs mátrix segítségével lehetséges. $(PAP^T)(Px) = (Pb)$

Lehetséges párhuzamosítások:

Amíg a Gauss-Seidel algoritmus sûrû mátrixokra szekvenciális, lehetséges azonosítani ritka mátrix partíciókat amikben nincs közös adat, tehát a számítás mûködhet párhuzamosan, és közben a szigorú precedencia szabályait is megtartjuk a Gauss-Seidel módszernek. Tehát egy egész ritka mátrix partíció számolható párhuzamosan mindenfajta kommunikáció nélkül. Blokk diagonális felbontást használunk a párhuzamosításhoz. Az utolsó diagonális blokk reprezentálja az osszekapcsolódási struktúrát az egyenleten belül, ami párosítja a particiókat a mátrix blokk diagonális részébe. Az egyenleteket gráffal reprezentáljuk amit színezni tudunk. A különálló színek sorokat reprezentálnak ahol x^{k+1} párhuzamosan számolható, mert egy színen belül nincs két csúcs között él. Definiáljuk a lineáris egyenletrendszer egy particionálását, ahol a P permutációs mátrix rendezi a mátrixunkat blokk diagonális formába.

$$PAP^T := \begin{bmatrix} A_{1,1} & 0 & ... & A_{1,m} \\ 0 & ... & ... & ... \\ ... & ... & A_{m-1,m-1} & A_{m-1,m} \\ A_{m,1} & ... & A_{m,m-1} & A_{m,m} \end{bmatrix}$$

Ha a blokk diagonál mátrix particiók $A_{i,i}, A_{m,i}$, és $A_{i,m}(1 \leq i \leq m-1)$ ugyan arra a processzorra kerülnek, akkor nincs kommunikáció amíg x_m^{k+1} kiértékelõdik. Kell egy módszer amivel ilyen blokk diagonális alak elérhetõ. Erre a csúcs-szakító (node-tearing) algoritmust használjuk aminek van egy felhasználó által megadható bemeneti maramétere, max_{DB}, a diagonális blokkok

maximális mérete. E paraméter változtatása nagyban befolyásolja a párhuzamos algoritmusunk teljesítményét.

A párhuzamos algoritmus a következõ részekbõl áll:

1. $x^{(k+1)}$ kiszámítása a diagonális blokkokban
2. $\widehat{b}$ kiszámítása a (mátrix $\times$ vektor) párhuzamos elõállításai által
3. $\widehat{x}^{(k+1)}$ kiszámítása az utolsó diagonális blokkban.
4. konvergencia ellenõrzése

Tapasztaliti eredmények:

A párhuzamos Gauss-Seidel algoritmus teljesítménye két összetevõtõl függ nagyon. Az egyik a mátrix rendezés teljesítménye az elõfeldolgozási fázisban, a másik pedig a párhuzamos Gauss-Seidel implementálásátálása. Elõször mindenféle sebesség méréseket vegeztek 3 nagy kiterjedésû mátrixon. Ezeket a mátrixokat egy szekvenciális program segítségével rendezték, és elõállították az adatstruktúrát a párhuzamos algoritmus számára. Ezen elõfeldolgozás során többféle max_{DB} (a blokk diagonális mátrixá alakító algoritmus bemenõ paramétere) értéket használtak. A tapasztalati eredményeket egy NPAC CM-5 nevezetû gép segítségével szerezték (1-32 processzort használva). Az 5. ábra mutatja ezt a relatív sebességet aszerint, hogy 2,4,8,16 vagy 32 processzort használtak. Látható, hogy logaritmikus fv-hez hasonló az egyes bemenõ mátrixokhoz a gyorsulás függvénye. Az EPRI-6K nevezetû mátrixnál a legjobb a teljesítménye az algoritmusnak, a másik 2 mátrixra körülbelul azonos. Ez annak a következménye, hogy a teljes számítási kapacitás 5% -a az utólsó diagonális blokk kiszámítása, mivel itt sok kommunikáció történik a többi folyamattal.

A relatív sebesség részletes vizsgálata található a 6. ábrán az EPRI-6K adatra. Azt mutatja, hogyan változik a gyorsaság a partíciók számának függvényében a blokk diagonális formára hozó algoritmusban. Jól mutatja a terhelesi egyensúlyhiányt ez a grafikon. A 4 bemenõ adatra 16 processzor hasznélatáig körülbelül azonosak a sebességek, azonban 32 processzornal szétválik és látható, hogy tobb partició esetén rosszabb a sebesség. Valamint vizsgáljuk meg a sebességet az algoritmusunk 4 külõnbozõ szakaszában. A 7. ábrán lévõ 4 diagramm ezeket az eseteket mutatja be, annak függvényében, hogy mekkora volt a max_{DB} érték (4 különbõzõ értékre vizsgálták: 128, 192, 256, 320). Logikus elvárás, hogy a processzorszám növelésével a lehetõ legjobb idõt érjük el , azaz a függvények egy vonal mentén folyamatosan csõkkenjenek. Ez a feltevés a mérési eredményeket nézve nagyrészt igaz, kivéve a utolsó blokk számításánál, ahol nagy mértékben megugrik a számítási idõ a processzorszám duplázása közben (ez jól látszik 8 processzor után). Ennek egy oka, hogy egyes mátrix rendezésekre nem tudja elosztani a munkát egyenletesen nagy számú processzorok esetén. Ez fõleg akkor fordul elõ ha a max_{DB} értéket túl nagyra választjuk. Kisebb max_{DB} értéknél még 60 vagy több processzorra is jobb sebesség érhetõ el.

Ismét elmondanám, hogy minden elérhetõ párhuzamosítás ebben a munkában a mátrixrendezés eredménye és felismerni az összefüggést a mátrix struktúrán belüli kapcsolódási mintában. Azon mátrixok amikben a teszteket végezték elég rendhagyóak és speciálisak, a legritkább mátrixok közé tartoznak ilyen szempontból. A csúcsok 84,4% -ának 3 vagy kevesebb éle van az EPRI-6K adatban. Következésképpen ezek a mátrixok jelentõs kihívást adnak arra, hogy

elõállítsunk hatékony párhuzamos algoritmusokat ritka mátrixokra. A 8. ábrán látható egy tipikus rendezése az EPRI-6K -nak ahol a max_{DB} egyenlõ 256 csúccsal. Ez a mátrix reprezentálja a hálózati gráf szomszédsági struktúráját és világosan mutatja a ritkaságot. A nem 0 elemek pontokkal vannak feltüntetve.

Következtetés

Az cikkben bemutatott párhuzamos Gauss-Seidel algoritmus az említett ritka és rendhagyó mátrixok esetén jó relatív gyorsulást mutat. A blokk diagonális mátrix felbontás megkönnyítette az implementációt és a probléma dekomponálását könnyebben feldolgozható részekre. A már említett node-tearing algoritmus segítségével megoldható a mátrix rendezése ilyen mátrixok esetén, és csökkenthetõ az egymástól függõ egyenlet párok száma, amibõl az következik ,hogy a gráf-színezõ algoritmusnak általában elég 2 vagy 3 szín az utolsó blokk színezésénél. A mátrixelemek közötti aktuális kapcsolatok meghatározása után lehetett bármiféle párhuzamosítást végezni (minél kevesebb egymás közötti kommunikációval járjon a processzorok munkája, és akkor nagy gyorsulás érhetõ el), ez derult ki a cikkbõl az elemzés során. Tehát lehet érezni, hogy a lehetséges párhuzamosítások korlátozottak. Megmutatták azt is, hogy relatív hatékonyság csökkenés érhetõ el , persze a mátrixtól függõen, ha 8 processzornál többet használunk futás közben.

Mivel elég speciálisnak kell lennie a bemenetnek ahhoz, hogy az algoritmus utolsó részében is hatékony gyorsulást tudjunk elérni a processzorszám növelésével, a következõ elemzés az algoritmus elsõ részének vizsgálatára szorítkozik. A teljes algoritmus kódja megtalálható a mellékelt cikkben. Nézzük az elsõ részét, $x^{(k+1)}$ kiszámítása a diagonális blokkokban:

```
for minden i-edik sorra ezen a processzoron
  x'_i <- x_i
  x_i <- b_i
  for j eleme [1,n] ahol a_ij nem egyenlõ 0
    x_i <- x_i - (a_ij * x_j)
  endfor
  x_i <- x_i / aii
endfor
```

Ahhoz, hogy a fenti problémát megoldjuk szükségünk van egy alkalmas párhuzamos környezetre amiben lehetõség van több processzor között elosztani a munkát, és így a futási idõt nagymértékben csökkenteni tudjuk. Ilyen lehetõséget nyújtanak a PVM vagy MPI programok, amiket egy daemon segítségével egyszerre több processzoron vagyunmk képesek futtatni. Azonban egy sokkal érdekesebb és napjainkban egyre nagyobb teret kapó rendszert választottam a szimulációhoz, ami a GRID. A többszintû párhuzamosság érhetõ el a GRID rendszerekben. Meghatározhatom, hogy mely erõforrásokat szeretném igénybe venni, persze csak azok közül amelyekre jogosúltságom van.

A szimulációmat a HUNGRID -ben végeztem, amibe az ELTE-s portal segítségével jelentkeztem be: http://pgrade.inf.elte.hu címen érhetõ el. Workflow szerkezetû programokat képes futtatni a GRID-ben a portal. Ez a egy irányított körmentes gráf amiben az egyes részprogramok (job) elindulása függ az õt megelõzõektõl, ha valami adatot vár amire szüksége van. Több ilyen gráfra végezteme futási eredmény vizsgálatot. Az eltérések ezek között a számított

mátrixok mérete, és ennek megfelelõen a gráf csúcsainak száma. Ezen méreteket a már fentebb említett max_{DB} érték határozza meg. A GRID-ben egy ilyen workflow elindítása után csak percekkel késõbb kezdenek el futni a csúcsokban definiált programok, ennek oka a sok elõzetes kommunikációban, a terheltségben és a jogosultságok vizsgálatában keresendõ. Azonban ha egy program több óráig is fut, ez a pár perc a futási idõben elenyészõ. Az elsõ mérési eredmény pontosan erre vonatkozik. Ha a max_{DB} túl nagy a diagonális blokkok kicsik lesznek. Ezeken a kis részmátrixokon végzett számítások hamar befejezõdnek. Itt a programok elindulásához szükséges idõ nagyobb a futási idõnél. Ezután azt vizsgáltam, hogy a mátrixok méretének növekedése hogyan befolyásolja a futási idõt.

Az egyszerû elemzés érdekében $n \times n$ -es mátrixokat vizsgáltam. A futási idõk arányai érdekesek számunkra, ezért tényleges mértékegységet nem adok sehol, csak nagyságrendekkel próbálom érzékeltetni az eltelt idõt. A futási idõ nem változott nagy mértékben ha n értéke 1000 alatt volt. Az elõkészületek a futáshoz nagyságrendileg megegyeztek, mivel az adatokat elküldeni a processzorhoz körülbelül ugyan annyi idõbe telt tetszõleges n esetén, valamint az eredmény visszaérkezése is hasonló idõt vett igénybe. Ez a teljes futási idõ 5, ha $n \leq 1000$, egy processzoron. De az n növelésével a futási idõ nagyban nõtt. $n = 1700$ -ra már 7, $n = 2400$ -ra 17, $n = 3100$ -ra körülbelül 50 -ra változott a futási idõ. Fontos kihangsúlyozni, hogy ezek a futási idõk egy processzorra vonatkoznak. Jól látható ebból, hogy a futási idõ n-hez (amit lineárisan növelek) nézve sokkal gyorsabban nõ (mélyreható számításokat nem végeztem, de talán mopndhatom azt, hogy exponenciálisan nõ). Azonban ha egymas mellett több ilyen szamítast is elindítok, ezek egymassal parhuzamosan tudnak futni a GRID-ben. Tehat a futasi idõ azonos lesz ha 2, 3 vagy m db ilyen számítást indítok el párhuzamosan, mert nem várnak egymástól adatot. Persze függ a gridben lévõ erõforrások terheltségétõl, de pl. ha 10 db program fut amely mindegyike $n = 2400$ akkor a futási idõ 17 lesz. Itt tudunk elérni nagyfokú párhuzamosítást.

Ha van kezdetben egy mátrix ahol $n = 10000$, ezt blokkdiagonális formára hozva elérjük azt, hogy sok kis részmátrixon kell számításokat végezni, amiket ráadásul lehet párhuzamosan is futtatni. Pl. ha 4 darab diagonális blokkunk van akkor a mellékelt ábrán lévõ workflow fut, amiben a négy egymás mellett lévõ csúcs egyszerre képes futni. Tehát a következtetés az, hogy ha megvan a blokkdiagonális felbontás, párhuzamosítással elég nagy gyorsítása érhetõ el az algoritmusnak. Az egész elemzés és szimuláció arra irányult, hogy bebizonyítsam, a GRID teljesen alkalmas az adott probléma megoldására, valamint a SZTAKI által fejlesztett portál segítségével egyszerûen tudjuk futtatni párhuzamosan a programjainkat, illetve a fent említett problémára adott programot.