SAT probléma kielégíthetőségének vizsgálata

masszív parallel

mesterséges neurális hálózat alkalmazásával

Tajti Tibor, Bíró Csaba, Kusper Gábor
{gkusper, birocs, tajti}@aries.ektf.hu

Eszterházy Károly Főiskola
Matematikai és Informatikai Intézet

Absztrakt

A mesterséges neurális hálózatok számos esetben bizonyítják alkalmazhatóságukat osztályozási problémák megoldására. Kutatásunk célja, hogy a SAT problémák vizsgálatára való alkalmasságukat vizsgáljuk, ezzel a SAT problémák területén és a mesterséges neurális hálózatok területén is tapasztalatokat szerezve. Mivel a nehezebb SAT problémák megoldásához nyilvánvalóan nagyobb intelligencia szükséges, és ennek teljesítményigénye is nagy, ezért a kutatás során futtatandó algoritmusok párhuzamosítása célszerű. Folyamatban levő kutatásunk tehát a párhuzamosítás adta lehetőségek tesztelése is egyben. A párhuzamosítás lehetősége különböző architektúrákon és szinteken is adódik, az SMP gépektől a GPU-k alkalmazásán keresztül a HPC –ig, beleértve a masszív parallel architektúrákat is.

SAT probléma

A logikai kifejezések kielégíthetőségének vizsgálata, kielégíthetőség esetén pedig egy megoldás vagy az összes megoldás megtalálása fontos és nehéz feladat.
A 0. rendű logikai kifejezések egy könnyen vizsgálható formája a konjunktív normálforma (KNF, angolul CNF).
A KNF forma azt jelenti, hogy logikai változók pozitív vagy negált alakjainak logikai VAGY művelettel összekapcsolt klózait logikai ÉS művelettel kötjük össze.
Példa:

(a ∨ c) ∧
(b ∨ c) ∧
(¬a ∨ b ∨¬c)

A SAT probléma NP-teljes, azaz nem ismert polinomiális idejű algoritmus, amely megoldaná.

Mivel a SAT probléma fontos többek között a logika, a mesterséges intelligencia és az áramkör tervezés területén, ezért nagy erőkkel keresik, hogyan lehet ezt a problémát kezelni.

Két megközelítés létezik:

- olyan SAT megoldó algoritmusok fejlesztése, amelyek valószínűleg polinomiális időben lefutnak,
- olyan speciális SAT részosztályok keresése, amelyek polinomiális időben megoldhatók.

Segítséget adhat még a megoldás sejtése. SAT ill. UNSAT sejtés esetén más solver-ekkel próbálkozhatunk.

Neurális hálózat alkalmazása

Mivel osztályozási feladattal állunk szemben, ezért felmerülhet a mesterséges neurális hálózatok alkalmazása a becslésre.
Amellett, hogy ez a mesterséges intelligencia technika alkalmazható lehet a SAT problémamegoldásban, a vizsgálatok tapasztalatot jelentenek mind a SAT, mind pedig a neurális hálózat területeken.

Számos neurális hálózat technika ismert.
Például:

- Perceptron
- Error Backpropagation
- Support Vector Machine
- Kohonen Self Organizing Map

A neurális hálózatok fő tulajdonságai közé tartozik, hogy képes mintaadatokból tanulni. Ez nagyon hasznos olyankor, amikor nem tudjuk vagy nem akarjuk algoritmizálni egy feladat megoldását.

A neurális hálózatok egyik fő alkalmazási területe az osztályozás, azaz neurális hálózatot betaníthatunk annak eldöntésére, hogy egy adott bemenet beletartozik-e egy osztályba, vagy sem.
Jelen esetben a kielégíthetőséget vizsgáljuk, másként azt, hogy egy probléma SAT vagy UNSAT.
Működhet-e a neurális hálózat által az osztályozás egy olyan problémakörre, amely NP-teljes?
Elvileg igen, sok minta input adatunk van, osztályozzuk őket, minden probléma vizsgálandó, hogy SAT -e.

Példa feladatok, CNF fájl minták:
Példa 1 változós SAT:

p cnf 1 2

-1 0

-1 0

Példa 1 változós UNSAT:

p cnf 1 2

1 0

-1 0

Példa kicsit nagyobb CNF-re:

p cnf 50 218

-37 -17 -44 0

-35 19 -2 0

-10 -34 -11 0

30 28 -40 0

15 28 -12 0

-36 -8 -48 0

13 -28 14 0

-14 17 25 0

… 218 klózig

Teljesítmény igény

A különböző nehézségű, méretű SAT problémák vizsgálatához készítettünk egy SAT probléma generátort.
Ez azért szükséges, mert a SAT problémák nehézsége változó, emiatt a neurális hálózat performanciaigénye is.

Nézzünk egy példát:

> 3SAT probléma 50 literállal 218 klózzal
>
> Input:
>
> 50 x 218 = 10900
>
> Esetleg
>
> 3 x 218 = 654

A sok bemenethez még több minta kell. Sok-sok minta...
Tapasztalat:

Nagy feladathoz nagyon sok minta kell, egyébként csak a mintát tanulja meg jól, a további adatokra pedig nem lesz hatékony.

Párhuzamosítás

Egy-egy tanulási kísérlet másodpercektől vagy percektől órákig terjed, ezért a feladat elvégzéséhez: nagy kapacitásra és párhuzamosításra van szükség.

Architektúrák
:

- HPC
- SMP
- MPP
- Vektorprocesszor
- Fürt – vastag, vékony
- GPU
- FPGA

A technikai fejlődés szolgáltatja a technikai lehetőségeket a párhuzamos futtatáshoz.
Kérdés, hogy milyen párhuzamosítási lehetőségeket találunk a neurális hálózat algoritmusok esetén.
Szerencsére több szinten is vannak lehetőségeink:
A neurális hálózat algoritmus összetett algoritmus, sok adatot dolgoz fel, és vannak független részfeladatok, amelyek párhuzamosan végrehajthatók (külön szálon számolni az egyes neuronokra eső részt, hiba visszaterjesztésnél független ciklusok).
Mivel sok műveletet vektorokon végez, ezért nagymértékű gyorsítás érhető el a vektorműveletek

párhuzamos végrehajtásával vektor processzoron vagy GPU-n.
A fentieken túl nem elvetendő a sok kutatási programfuttatás olyan szervezése, hogy egymástól teljesen vagy nagy mértékben független különálló programokat futtatunk párhuzamosan.

A lehetőségek közti döntést komoly megfontolásnak kell megelőznie. A párhuzamosan végrehajtandó feladatok csak akkor járnak jó eredménnyel, ha a szinkronizáció nem okoz több veszteséget, mint amennyit a párhuzamosításon nyerünk.

Sokszor tapasztaltuk, hogy klaszteren futtatott párhuzamos program futási ideje több volt, mint ugyanannak a programnak a nem párhuzamos változatáé egy gépen.

Nézzünk példát a vektorműveletek párhuzamosítására és a szinkronizáció költségére.
Tesztünket GPU-n futtatjuk, az eredményt összevetjük a CPU-n való nem párhuzamos futás időeredményével.

A szinkronizáció ára

GPU kernel vektorok összeadására (python opencl használatával):

```
kernel = """
__kernel void vector_add(__global float* A,
__global float* B,
__global float* C)
{
// each kernel instance has a different global id
int i = get_global_id(0);
C[i] = A[i] + B[i];
}
"""
```

Mérhető időtartamhoz 1 millió elemű vektorokat adunk össze, és ezt 1000-szer futtatjuk.
Itt nincs számottevő szinkronizáció.
Ennek köszönhetően a GPU-n nagy hatékonysággal fut a program.

Futási idők:
GPU -n – 0.0254 s
CPU -n – 16.2 s

Most nézzünk példát olyan műveletre, ahol már jelentősebb szinkronizáció szükséges. A vektorok belső szorzata esetén a páronkénti szorzat gyors elkészítése után még szükség van egy időigényes szinkronizációra, mégpedig a szorzatok összegzésére.

A python opencl-nek az ilyen szinkronizációhoz kész kernel függvénye van:

```
dot =  ReductionKernel(
            ctx, dtype_out=numpy.float32, neutral="0",
            reduce_expr="a+b", map_expr="x[i]*y[i]",
```

```
                arguments=" __global const float *x, __global const float
    *y"
        );
```

A mért eredmények:
GPU -n – 3.48 s
CPU -n – 18.0 s

Összegzés

Mint az előző példából láthattuk, még egy számítógépen, sőt egy GPU-n belül is jelentős mértékű a párhuzamosítás költsége, megközelítheti, sőt túl is lépheti a párhuzamosítás adta nyereséget.
MPP és klaszter architektúra esetén ez fokozottan így van.
Ilyen esetekben cél tehát a párhuzamosítás hatékonyságának növelése, minél kisebb szinkronizációs költség elérése.
Kutatásaink során a leghatékonyabb párhuzamosítást a legegyszerűbb technikával értük el, mégpedig a független feladatok párhuzamos futtatásával.

Források

- Kusper Gábor, Kovásznai Gergely, Bíró Csaba: A SAT probléma különböző reprezentációinak vizsgálata oktatási szempontból, AgriaMédia 2008 konferencia, 2008.
- Andreas Klöckner: GPU Programming in Python with PyOpenCL and PyCUDA, 2012