Szakdolgozat

Nemegyensúlyi rendszerek vizsgálata grafikus kártyás szuperszámítógéppel

Domokos Zoltán

Fizika BSc., Alkalmazott fizikus szakirány

III. évfolyam

Témavezető:
Ódor Géza
tudományos tanácsadó

Belső konzulens:
Jánosi Imre
egyetemi docens

Eötvös Loránd Tudományegyetem
MTA TTK Műszaki Fizikai és Anyagtudományi Intézet

2013

Abstract in English

The Susceptible-Infected-Susceptible (SIS) model is the simplest model for information and epidemic spreading. My thesys focuses on the spectral analysis of the SIS process in Barabási-Albert scale free networks and the opportunities to exploit the nVIDIA graphics processing units' parallel processing capabilities in the spectral decomposition analysis of the SIS model. I found that some important calculations of the method can be implemented on the GPU. Demonstrating the abilities of this program, I show that for network sizes larger than $2 \cdot 10^5$, in case of initial seed sizes ≥ 30, the power law degree distribution does not fit the numerical results, which is in agreement with recent studies, and might have an influence on the controll parameter in the SIS model. A set of measurements for the Inverse Participation Ratio has also been made to validate the code.

Kivonat

A Susceptible-Infected-Susceptible (SIS) modell a legegyszerűbb információ és betegség terjedést leíró modell. Szakdolgozatomban a SIS folyamat spektrális dekompozíción alapuló vizsgálatával foglalkoztam Barabási-Albert hálózatokon, és lehetőségeket kerestem arra, hogy a rendelkezésemre álló nVIDIA grafikus feldolgozó egységek párhuzamos számítási kapacitásait is kihasználjam. A program képességeinek demonstrációja céljából megmutatom, hogy $2 \cdot 10^5$ eleműnél nagyobb hálózatok esetén, ha a kezdeti mag mérete ≥ 30, a hatványfüggvény szerint haladó fokszámeloszlás nem érvényes, ami összhangban van a legutóbbi kutatási eredményekkel, és érintheti az SIS modell kontroll paraméterének alakulását. A program működőképességét egy Inverz Participációs Rátára vonatkozó méréssorozattal teszteltem.

Köszönetnyilvánítás

Ezúton is szeretném megköszönni Ódor Gézának, hogy elvállalta a témavezetést. Ezenkívül köszönöm mindazoknak, akik elolvasták a készülő dolgozatot és megjegyzéseikkel segítették annak befejezését.

Budapest, 2013. május 31.

Tartalomjegyzék

Ábrák jegyzéke

1. Bevezetés

Napjaink fizikájában az egyik leggyorsabban fejlődő kutatási terület a hálózatok és a rajtuk értelmezett különféle dinamikai folyamatok tulajdonságainak feltérképezése. Ezen terület megjelenése a szélesebb tudományos köztudatban nagyrészt Barabási László és Albert Réka 2002-es munkájának köszönhető, melyben egyszerű hálózat növekedési modellt javasoltak a valódi skálafüggetlen hálózatok alapvető tulajdonságainak mesterséges rekonstrukciójára. Néhány magától értetődő társadalomtudományi és gazdaságtudományi alkalmazási lehetőségen túl a hálózatok elméletének születőfélben lévő diszciplínája a XXI. században meghatározó lehet a kognitív idegtudományok, az epidemiológia, és a számítógépes infrastruktúrák elméleti kutatása területén is. A legérdekesebb, egyúttal legkevésbé kézenfekvő tény azonban az, hogy a szilárdtestfizikában ez a megközelítsmód szintén új távlatokat nyitott az elmúlt években, és többen úgy vélik, a spin üvegek viselkedését, különféle rendezetlen rendszerek fázisátalakulásait, sőt a szupravezetés jelenségének bizonyos aspektusait is [1] hatékonyan lehet leírni a hálózatok elméletének - napjainkban még korántsem teljes - eszköztára által.

Szakdolgozatomban igyekeztem megvizsgálni egy már kidolgozott módszer számítógépes implementációjának lehetőségét, továbbá arra kerestem a választ, hogy lehetséges e bizonyos részletszámításokat vagy a számítások egészét a számítógép grafikus feldolgozó egységén végezni. A grafikus kártyás számítások szintén felfutóban lévő, népszerű kutatási területet jelentenek, azonban a GPU hálózatok modellezésében való felhasználására tett kísérletek eddig kevés kivételtől eltekintve elkerülték a kutatók figyelmét. Ennek oka részben az, hogy a terület új, ezért a tudományos közösség mindeddig jórészt a masszívan paralell SIMD architektúrához legjobban illeszkedő problémákra fókuszált, és a gyakran használt lineáris algebrai rutinok optimalizálása is kellően komoly kihívást jelentett ahhoz, hogy ezekre irányuljon a figyelem.

Magam azt a célt tűztem ki ezen szakdolgozat keretében, hogy megvizsgáljam a szakirodalomban fellelhető kísérleteket, amelyek a Barabási-féle gráfok grafikus kártyás előállítását célozták, és ezek alapján elkészítsek egy olyan alkalmazást, amely témavezetőm segítségére lehet az MTA-MFA-ban végzett kutatásai közben. Ezt a célkitűzést részben sikerült is megvalósítani, miközben értékes, más területen is könnyen hasznosítható ismeretekre tettem szert a CUDA programozási modellel, és

a különféle lineáris algebrai és ritka-lineáris algebrai könyvtárak használatával kapcsolatban. Igyekeztem az eredeti Barabási -féle algoritmust követni, mivel ekkor az ismert analitikus megoldással az eredmények könnyen összevethetők, egyszerűsítve a tesztelés fázisát.

Munkám során a Susceptible-Infected-Susceptible (SIS) folyamatot vizsgáltam Barabási-Albert-féle preferált csatolású hálózatokon. Kiderült azonban, hogy az egyszerű fák szomszédsági mátrixainak adottságai miatt az általam használt sajátérték megoldó algoritmus gyakran nem konvergál, így eredményeket a fák kapcsán nem prezentálok. Később bővebben lesz szó a SIS modellről, ezért itt nem részletezem, egyelőre elegendő elöljáróban annyit megemlítenem vele kapcsolatban, hogy ez egy egyszerű, információ vagy fertőzés - általánosan fogalmazva aktivitás - terjedésének dinamikáját leíró modell. Érdeklődésem középpontjában az állt, hogy miként lehetne a hosszú idő alatt beálló állandósult állapotban meghatározni a rendszerben maradó információ átlagát mint rendparamétert, hogyan lehetne jellemezni az állandósult állapotot, illetve miként lehet megfigyelni ritka régió effektusokat. Mindezt próbáltam a grafikus kártya nyújtotta előnyök lehető legjobb kihasználásával megtenni.

A dolgozat tagolásában igyekeztem elegendően nagy teret adni az elméleti háttérrel kapcsolatos alapoknak, és a programom bemutatásának is. Remélhetőleg, azon olvasók számára is könnyű lesz végigjutni a szövegen, akik nem mozognak otthonosan a témában, vagy esetleg érdeklődő alsóbb éves hallgatók. Gyakorlottabb olvasók a túlzottan didaktikusnak ható magyarázatokat a megfelelő elkülönítésnek köszönhetően könnyen átugorhatják. Azok, akik már találkoztak a spektrális dekompozíciós analízissel, a SIS modellel, a Barabási-Albert hálózatok generáló algoritmusával, ezen részekkel való időtöltés helyett rögtön rátérhetnek a program bemutatására és a futtatások során kapott eredmények értelmezésére. Ezen megfontolásokból kifolyólag az első fejezetben kerül sor a SIS modell rövid felvázolására, amit a spektrális dekompozíciós módszer kicsit részletesebb tárgyalása követ. A dolgozat második része a program megvalósításáról, a megvalósítás folyamán felmerült problémák kezeléséről szól. Ezzel már részben el is jutunk az utolsó fejezet témájához, ami a program futtatása során kapott eredmények értékelése.

Az utolsó fejezetben tehát szó esik arról is, hogy a program használatával milyen eredmények születtek. Ez azért fontos, mert témavezetőm korábban a népszerű Octave programmal végzett számításokat, amik időigényességük miatt erősen korlátoz-

ták a számoláshoz generált mesterséges hálózat méretét, azaz a gráf csomópontjainak számát. Így kifejezetten érdekes volt látni, hogy nagyobb hálózatoknál is jó egyezést mutatnak a numerikus számítások és az analitikus módszerek, továbbá sikerült kimutatni, hogy a hálózat generálás kezdeti feltételeinek beállítása miként befolyásolja a hálózat szerkezetét, és ezzel együtt a kvantitatív elemzés végeredményeként kapott mennyiségeket is.

2. Módszertani alapok

2.1. A Susceptible-Infected-Susceptible modell

A Susceptible-Infected-Susceptible modell az egyik legegyszerűbb olyan modell, amely a tetszőleges topológiájú hálózatokon történő információ terjedést kívánja leírni. Az ilyen, rövidített formában SIS-nek nevezett, folyamatban a gráf csúcsai kétféle állapotban lehetnek. Egyik állapot az, amikor a csúcs aktív. Epidemiológiai alkalmazás esetén ekkor azt mondhatnánk, hogy az adott csúcs fertőzött. Természetesen a másik eset az inaktív állapot, ami annak felel meg, hogy a csúcs egészséges. Ha egy tetszőleges csúcs a folyamat során mindkét állapotban lehet, márpedig a SIS esetén ez a helyzet, akkor meg kell adni azt is, hogy az átmenet miként következhet be. A modell feltevései szerint minden pillanatban a fertőzött csomópontok λ rátával továbbadják a fertőzést szomszédaiknak vagy egységnyi rátával meggyógyulnak (inaktiválódnak). Ez azt jelenti, hogy a fertőzés továbbadásának valószínűsége egy szomszédnak mindig $\frac{\lambda}{1+\lambda}$, és a gyógyulás valószínűsége $\frac{1}{1+\lambda}$ [2]. Quenched Mean Field közelítésben, ami az átlagtér elmélet (Mean Field Theory) időben nagyon lassan változó vagy változatlan hálózatokra vonatkozó változata [3], a hálózat topológiájának ismeretében könnyen felírható a SIS folyamat ráta-egyenlete, amelyből a későbbi számítások menete megadható, a vizsgált mennyiségek származtathatók.

Általános hálózat esetén, a spektrális felbontáson alapuló vizsgálat során, amint a bevezetőben már elhangzott, arra keressük a választ, hogy létezik-e olyan paraméter, amelynek kritikus értéke meghatározza, hogy állandósult állapotban az egyes hálózati csomópontok aktivációs valószínűsége minden csomópontra nulla e vagy sem. Amennyiben találhatunk ilyen paramétert, annak kritikus értéke alapján előre jelezhető, hogy a fertőzés hosszú távon kihal a rendszerből, amit a hálózat testesít meg, vagy tartósan benne marad, és hosszú távú jelenséggé válik. Ez az

úgynevezett kontroll paraméter a SIS modellben a λ fertőzési ráta, a kritikus λ felett tehát a fertőzöttség állandósult valószínűsége a csomópontokra nem nulla, míg alatta igen. Ha a QMF feltételezéseiből indulunk ki, akkor figyelembe lehet venni annak a hálózatnak a topológiáját is, amelyben a SIS folyamat végbe megy.

Heterogén átlagtér elmélet alkalmazása esetén erre nincs mód, de ez utóbbi esetben egy kevésbé részletgazdag leírást kapnánk, ami általában akkor célravezető, amikor a folyamatnak teret adó hálózat topológiája időben változó, továbbá ez a változás nehezen nyomon követhető. Feltételezzük, hogy a folyamat egy időben változatlan elrendezésű, vagy csak nagyon lassan változó hálózaton megy végbe, sem az élek súlyai, sem a szomszédsági viszonyok nem rendeződnek át, tehát a QMF megközelítés alkalmazásának feltételei adottak. Meg kell jegyeznem azonban, hogy a modell dinamikus korrelációinak fluktuációit elhanyagoljuk az átlagtér közelítésekben, így a QMF-ben is.

2.2. A spektrális dekompozíciós módszer elvi alapjai

Ebben az alfejezetben megmutatom hogy miként lehet a programomban számolt outputok segítségével értékes mennyiségeket meghatározni. Ezt [4], [5] szerzők cikkei alapján mutatom be.

Ráta-egyenletünk felírásához definiálni kell az állapottér elemeit. Hálózatunknak ha N csomópontja van, akkor vehetünk olyan N dimenziós vektortérbeli elemeket, azaz vektorokat, amelyeknek algebrai reprezentációjában az i-ik komponense annak valószínűségét fejezi ki, hogy az i-ik csomópont t időpillanatban éppen aktív. Az állapottér elemei ezek a vektorok lesznek, a későbbiekben jelöljük őket „ρ_i”-vel.
Felírhatjuk a ráta-egyenletet véges Δt időintervallumra:

$$\underline{\boldsymbol{\rho}}\,(t+\Delta t) - \underline{\boldsymbol{\rho}}\,(t) = -\underline{\boldsymbol{\rho}}\,(t)\,\Delta t + \left(\underline{\mathbf{1}} - \underline{\boldsymbol{\rho}}\,(t)\right)\cdot\lambda\cdot\underline{\underline{\boldsymbol{A}}}\cdot\underline{\boldsymbol{\rho}}\,(t)\,\Delta t \tag{2.1}$$

Eszerint annak valószínűsége, hogy egy csomópont fertőzött t-ről $t+\Delta t$-re úgy változhat meg, hogy a csomópont fertőzött állapotából meggyógyul, ezt fejezi ki a jobb oldal első tagja, vagy egészséges állapotából, egy fertőzött szomszéd megfertőzi. Mostantól a vektoregyenlet alak helyett az egyes vektorok komponenseire vonatkozó

egyenleteket írom ki, ahol az i index a vektor komponensének indexét fogja jelenteni:

$$\rho_i(t+\Delta t) - \rho_i(t) = -\rho_i(t)\,\Delta t + (1-\rho_i(t)) \sum_{j\neq i,\, j=1}^{N} A_{ij}\lambda\rho_j(t)\,\Delta t \tag{2.2}$$

Átrendezve adódik:

$$\frac{\rho_i(t+\Delta t) - \rho_i(t)}{\Delta t} = -\rho_i(t) + (1-\rho_i(t)) \sum_{j\neq i,\, j=1}^{N} A_{ij}\lambda\rho_j(t) \tag{2.3}$$

Ahol az A_{ij} nem más, mint a hálózat gráfjának szomszédsági mátrixa. Érthető okokból a fertőzöttség valószínűsége az idő t-ről $t+dt$-re változása alatt úgy növekszik, hogy a növekmény arányos a λ fertőzési ráta és a j indexű szomszéd fertőzöttségi valószínűségének szorzatával. Az A_{ij} tag pedig a szomszédság tényének kifejezője, ugyanis értéke 1, ha az i és j indexű elemek szomszédok, és nulla, ha nem. Később az A_{ij} mátrixelemek előállításáról bővebben lesz szó. Ebből látszik, hogy a fertőzés valószínűsége azon csúcsok esetén növekszik nagyobb mértékben, amelyeknek több nagy valószínűséggel fertőzött szomszédja van. A negatív előjellel vett ρ_i pedig azt fejezi ki, hogy a t pillanatban fertőzött csomópontok $t+dt$-re egységnyi rátával meggyógyulhatnak. Ha most elvégezzük a $\Delta t \to 0$ határátmenetet, a ráta-egyenlet megszokott differenciális alakját kapjuk:

$$\frac{d\rho_i(t)}{dt} = -\rho_i(t) + (1-\rho_i(t)) \sum_{j\neq i,\, j=1}^{N} A_{ij}\lambda\rho_j(t) \tag{2.4}$$

Könnyen belátható, hogy az állandósult állapotban, amikor $t \to \infty$, $\frac{d\rho_i}{dt} \to 0$, ezért az egyenlet bal oldalát tekinthetjük zérusnak, vagyis azt mondhatjuk, hogy állandósult állapotban a $\underline{\boldsymbol{\rho}}$ vektor időben változatlan lesz:

$$0 = -\rho_i(t) + (1-\rho_i(t)) \sum_{j\neq i,\, j=1}^{N} A_{ij}\lambda\rho_j(t) \tag{2.5}$$

Innen egyszerű átrendezéssel kapjuk a következő összefüggést a ρ vektorra:

$$\rho_i = \frac{\lambda \sum\limits_{j \neq i,\, j=1}^{N} A_{ij}\rho_j}{1 + \lambda \sum\limits_{j \neq i,\, j=1}^{N} A_{ij}\rho_j} \tag{2.6}$$

A [5] cikk szerint a differenciál-egyenlet rendszerek elméletéből ismert stabilitás analízis segítségével kimutatható, hogy létezik olyan λ_c kritikus érték, amelynél a ρ minden i-re nulla a $t \to \infty$ határesetben, ezért a λ_c meghatározásával bármely hálózat esetén meg tudjuk mondani, hogy a SIS folyamathoz hasonló dinamika szerint terjedő információ vagy vírus tartósan a rendszerben marad e vagy nem. Rendparaméterként a ρ_i várható értékét használhatjuk, amely nulla, ha λ_c alatti a fertőzési ráta, és véges érték, ha λ_c feletti fertőzési rátát alkalmazunk. Eddig csak annyit állítottunk, hogy a λ lesz a keresett paraméter, melynek kritikus értéke érdekes számunkra, de a kritikus érték analitikus meghatározásáról még nem tudunk semmit. Ebben lesz segítségünkre a spektrális dekompozíció módszere.

További lépést teszünk a λ_c-re vonatkozó analitikus összefüggés megadása felé, ha felismerjük, hogy a hálózat szomszédsági viszonyait kódoló A_{ij} mátrix valós, szimmetrikus, tehát a Perron-Frobenius tétel szerint sajátvektorai teljes ortonormált bázist alkotnak, amely bázis elemeinek lineáris kombinációjaként előállítható a $\underline{\boldsymbol{\rho}}$ vektor. [5]
Ekkor a $\underline{\boldsymbol{\rho}}$ vektorra a következő kifejezést írhatjuk fel:

$$\rho_i = \sum_{\Lambda=1}^{N} c\left(\Lambda\right) q_i\left(\Lambda\right) \tag{2.7}$$

Itt $c\left(\Lambda\right)$ jelöli az adott sajátvektorra vonatkozó koefficienst, Λ a sajátérték, $q_i\left(\Lambda\right)$ pedig a sajátvektor i-ik komponense. Ezt beírva abba az egyenletbe 2.6, melyet a ráta-egyenletből az egyensúly beálltát feltételezve kaptunk, a következő kifejezést kapjuk:

$$c\left(\Lambda\right) = \lambda \sum_{\Lambda'} \Lambda' c\left(\Lambda'\right) \sum_{i=1}^{N} \frac{q_i\left(\Lambda\right) q_i\left(\Lambda'\right)}{1 + \lambda \sum\limits_{\widetilde{\Lambda}=1}^{N} \widetilde{\Lambda} c\left(\widetilde{\Lambda}\right) q_i\left(\widetilde{\Lambda}\right)} \tag{2.8}$$

Ebből megmutatható, hogy a λ_c kritikus érték (QMF közelítésben) nem más, mint

a hálózat $\underline{\underline{\boldsymbol{A}}}$ szomszédsági mátrixának legnagyobb sajátértéke invertálva:

$$\lambda_c = \frac{1}{\Lambda_{max}} \tag{2.9}$$

Már csak az állandósult állapotvektorra van szükségünk, ami így áll elő:

$$\rho(\lambda) \approx \alpha_1 (\lambda\Lambda_1 - 1) + \alpha_2 (\lambda\Lambda_2 - 1)^2 + \dots \tag{2.10}$$

A közelítést a hivatkozott irodalomban [4] másodrendig írták fel, azonban a numerikus számítások során jól látszott, hogy a másod és harmadrendű tag igen ritkán játszik jelentős szerepet az Inverz Participációs Ráta (IPR) meghatározásában, ezért jobbára elhagyható. Fenti képletünkben α koefficiensek szerepelnek, amelyekről egyelőre nem esett szó. Ezeknek meghatározott számítási módja a következő:

$$\alpha_j = \frac{\sum_{i=1}^{N} q_i(\Lambda_j)}{N \sum_{i=1}^{N} q_i^3(\Lambda_j)} \tag{2.11}$$

Tehát a sajátvektorokra vonatkozó formula kifejtve a következő:

$$\rho(\lambda) \approx \frac{\sum_{i=1}^{N} q_i(\Lambda_1)}{N \sum_{i=1}^{N} q_i^3(\Lambda_1)} (\lambda\Lambda_1 - 1) + \frac{\sum_{i=1}^{N} q_i(\Lambda_2)}{N \sum_{i=1}^{N} q_i^3(\Lambda_2)} (\lambda\Lambda_2 - 1)^2 + \dots \tag{2.12}$$

Ha adott a kritikus λ, azzal még viszonylag kevés következtetésre nyílik lehetőségünk, mindössze annyit állapíthatunk meg, hogy a fertőzés az állandósult állapotban abszorbeálódik e vagy sem. Ezzel szemben a sajátvektorok felhasználásával 2.12 szerint előállított állandósult állapotvektor már egészen részletes leírását adja a SIS folyamat állandósult állapoti alakulásának.

Jelenlegi állapotában a program képes a 2.12 első rendű közelítésben szereplő összes érték meghatározására. Könnyű lenne úgy bővíteni hogy az állandósult állapotbeli $\rho(\lambda)$ másodrendű közelítését is kiszámítsa.

Elsőrendű közelítésben a legnagyobb sajátértékhez tartozó sajátvektor lehet lokalizált vagy delokalizált állandósult állapotban, az aktív fázisban, attól függően, hogy az aktivitás sűrű csomókban koncentrálódik, vagy egyenletesen terjed szét a

hálózatban. Az Inverz Participációs Ráta ezt mérhetővé teszi, számítási módja [6] a következő:

$$IPR\left(\Lambda\right) = \sum_{i=1}^{N} q_i^4\left(\Lambda\right) \tag{2.13}$$

Az IPR alacsony, nulla közeli értékénél a sajátvektor delokalizált, a fertőzés egyenletesebb eloszlású, míg a magas - egyhez közeli - érték a lokalizáció jele. Ódor Géza [4] munkájában arra világít rá, hogy az utóbbi esetben a magas IPR, tehát az állandósult állapoti lokalizáció a SIS folyamat időfejlődése során megjelenő ritka régiók miatt alakulhat ki. A ritka régiók olyan halmazok a hálózatban, melyek a hálózat többi részétől eltérő fázisban vannak, és lebomlásuk időben rendkívül lassú. Az IPR alapján ebből kifolyólag következtetni lehetne arra, hogy a SIS folyamat időfejlődése során a ritka régiók megjelenése befolyásolta az állandósult állapotot.

Most, hogy a QMF közelítésen alapuló, spektrális dekompozíciós analízist, és a vizsgálandó SIS folyamatot megismertük, bemutatom a numerikus számításokhoz használt programot. Összességében annyit érdemes róla tudni a részletesebb tárgyalás előtt, hogy az imént vázolt számítási lépések közül csupán a sajátértékek, sajátvektorok, a ρ_i állandósult állapotvektor, és az IPR számítását kell elvégeznie, a többi lépés az analitikus levezetés részét képezi, de egyébként nem járul hozzá az érdeklődésünkre számot tartó kimenet előállításához.

A program része az A_{ij} felépítése is, természetesen többféle gráf generáló algoritmust is implementálhattunk volna, azonban a választás a bevezetőben is említett súlyozott Barabási-Albert hálózatra esett. A program így két nagyobb egységből áll, az első egységet a gráf generálás képezi, amelynél a legnagyobb kihívást az jelentette, hogy a szomszédsági mátrixot rögtön CSR ritka-mátrix formában állítsuk elő, amiről később még lesz szó. A második egység a sajátérték számítás, amely a GPU-n történik, és a CPU-n végzett számításnál elvben gyorsabbnak kellett volna lenni, azonban ezt a célkitűzést csak részben sikerült megvalósítani.

3. A program ismertetése

A korábbi fejezetekben felvázolt, spektrális dekompozíción alapuló analízis segítségével szeretnénk meghatározni állandósult állapot esetére a λ fertőzési ráta kritikus értékét, és a ritka régió effektusok kialakulását, illetve az állapot lokalizációjának

mértékét jellemző Inverz Participációs Rátát (IPR). Mivel minket nem egy adott valós hálózat jellemzése érdekel, hanem általában bizonyos elméleti megfontolások alapján mesterségesen generált gráfok tulajdonságai, a programnak elsőként ezeket kell előállítania. Az előállítás azt jelenti, hogy a gráf szomszédsági mátrixát létrehozza a gráftípushoz tartozó algoritmus alapján, majd bevezeti az élekhez tartozó súlyokat, így egy bináris mátrix helyett egy olyan új mátrixot kapunk, amelynek elemei már nem csupán egyesek vagy nullák, hanem tetszőleges valós számok. Ezt követően kerül sor a mátrix első, és egyes esetekben második sajátértékének, illetve a hozzájuk tartozó sajátvektorok kiszámítására, amelyekből rendre a kritikus λ, valamint az IPR előállítható. Részletesen az egyes lépések az alábbiakban kerülnek leírásra. Elsőként nézzük meg, hogy az egyszerű (súlyozatlan) Barabási-Albert hálózat hogyan generálható!

3.1. A Barabási-Albert hálózatok generálása

Az iterációs lépések megkezdése előtt szükség lesz egy $N \times N$-es mátrixra (jelöljük $\underline{\underline{\boldsymbol{A}}}$-val), ahol N pontosan a folyamat végeredményeként kapott gráf elemszáma, és feltételezzük, hogy a csúcsok egytől N-ig terjedő indexhalmazzal rendelkeznek majd. Mátrixunk $A\,(i,j)$ eleme egy, ha az i és j indexű csúcsok szomszédok, egyébként nulla. A gráf élei nem irányítottak, nincs szükség arra, hogy az irányítást külön jelöljük, elegendő azt ábrázolni, hogy két csúcs között van e él vagy nincs. Tehát az $\underline{\underline{\boldsymbol{A}}}$ mátrix, ami nem más mint a szomszédsági mátrix, minden információt hordoz, amire csak szükségünk lehet ahhoz, hogy a gráf topológiáját ismerjük. Mivel a szomszédsági mátrix elemei jórészt nullák, ráadásul szimmetrikus is, megfelelő ritka mátrix formátum választásával jelentős mennyiségű számítási erőforrást (elsősorban memóriát) spórolhatunk meg. Ha az egyes élekhez súlyokat rendelünk, akkor a szomszédsági mátrix nem nulla elemeit megszorozzuk a megfelelő súlyfaktorral, így egy $\underline{\underline{\boldsymbol{B}}}$ mátrix az eredmény, amelynek sajátértékeit és sajátvektorait a kritikus λ és az IPR számításához használjuk majd. Súlyokat az eddig elkészült program nem vezet be az élekhez, de egy ilyen kiegészítés kiváló lehetőség képességeinek finomítására.

Mikor egyszerű Barabási-Albert fát állítunk elő, kezdetben megadunk egy M csúcsból álló gráfot, amelyben minden csúcsnak szomszédja az összes többi. Ehhez a gráfhoz az iterációs lépések során rendre hozzáadunk egy-egy újabb csúcsot, amely úgy kapcsolódik a gráfhoz, hogy a csúcsok közül adott valószínűséggel választunk

egyet, és ezzel az új csúcs létesít egy élt. A valószínűség, amely megadja, hogy mekkora eséllyel csatlakozik az adott csúcshoz az új, egyenesen arányos a gráf egyes csúcsainak fokszámával (lineáris preferenciális csatolás) [7]. A gráfot generáló algoritmus röviden így írható le:

- Kezdetben N_0 csúcs van, mind összekötve.
- Kiszámítjuk a kezdő csúcsok fokszámai alapján a csatlakozás π valószínűségét, ami $\sum \frac{k_i}{k_{ossz}}$.
- A $0-1$ intervallumot felosztjuk úgy, hogy a részintervallumok $0-M$-ig a csatlakozási valószínűségeknek megfelelő hosszúságúak legyenek.
- Véletlen számot generálunk a $0-1$ intervallumban, és megnézzük, hogy a megállapított intervallumok közül melyikbe esik.
- Az előző lépésben kijelölt intervallumnak megfelelő csúcshoz, és az új elemhez adunk egy élt, az említett két csúcsot szomszédként jelöljük meg.
- Ismételjük a lépéssorozatot $N - N_0$-szor.

A fenti folyamat kicsit másképp fest akkor, ha nem fa szerkezetű a hálózat, hanem megengedjük, hogy az új csomópontok egynél több éllel kapcsolódjanak be a hálózatba. Léteznek olyan preferenciális csatolású hálózatot generáló algoritmusok (pl. Herbert Simon algoritmusa [8]), amelyekben lehetőség van arra, hogy egy csúcs a gráfban több élt létesítsen ugyanazzal a szomszédjával, azonban ez a Barabási Albert hálózatoknál nem fordulhat elő [7]. Ebből következik, hogy a csatlakozó csúcs éleinek maximális száma nem lehet több a kezdeti gráfban elhelyezett elemek számánál. Feltétezzük továbbá a csatlakozó élek számának (m) konstans voltát. Például nem érkezhet új csúcs két éllel, ha a többiek hárommal csatlakoztak érkezésükkor. Több éllel történő beérkezés esetén az algoritmus módosítandó:

- Kezdetben N_0 csúcs van, mind összekötve, az új elemek m éllel fognak érkezni, $N_0 \geq m$.
- Bevezetünk egy új csúcsot a korábban ismertetett algoritmus szerint.
- Az előző lépésben kijelölt csúcshoz, és az új elemhez adunk egy élt, az említett két csúcsot szomszédként jelöljük meg.

- Újraszámoljuk a csatlakozási valószínűségeket, de ezúttal kihagyjuk azokat a csúcsokat, amelyekhez már adtunk ebben a körben élt. Ehhez ki kell számítani a fokszámok összegét úgy, hogy a kizárt csúcsok fokszámait nem vesszük figyelembe.

- Ismételjük az élek hozzáadását m-szer.

- Ha bevezettük az új csúcsot, azaz m szomszédhoz kötöttük, akkor az eljárás az előző esethez hasonlóan $N - N_0$-szor végzendő.

Mikor súlyozott Barabási Albert fát (a továbbiakban WBAT) készítünk, az A szomszédsági mátrix előállítása hasonlóan történik, tehát a gráf topológiája hasonló, de az élekhez egy meghatározott összefüggés szerint súlyokat rendelünk. A súlyok szerepéről már volt szó, ezért itt csak az lényeges, hogy az $\underline{\underline{\boldsymbol{A}}}$ és a $\underline{\underline{\boldsymbol{B}}}$ mátrix előállítása történhet úgy, hogy elsőként $\underline{\underline{\boldsymbol{A}}}$-t állítjuk elő, és csak ezután generáljuk $\underline{\underline{\boldsymbol{A}}}$ alapján $\underline{\underline{\boldsymbol{B}}}$-t a súlyok bevezetésével, hiszen $A_{ij} = \omega_{ij} B_{ij}$, ahol az ω_{ij} az i-dik és j-dik csomópontot összekötő él súlya. Mint később látni fogjuk, a gráf növekedésének szimulációja, azaz $\underline{\underline{\boldsymbol{A}}}$ előállítása az SIMD/SIMT architektúrához kevéssé illeszkedő probléma, ellenben az élek súlyainak kiszámítása, a $\underline{\underline{\boldsymbol{B}}}$ mátrix elemeinek számolása jól párhuzamosítható, ebből adódik az iménti állítás jelentősége.

Ha már ismert $\underline{\underline{\boldsymbol{B}}}$ (BAT modellnél $\underline{\underline{\boldsymbol{A}}} = \underline{\underline{\boldsymbol{B}}}$ a súlyozatlanság miatt), a skálafüggetlen hálózatokon zajló SIS folyamat spektrális analíziséről szóló részből ismert összefüggések alapján, ahhoz, hogy a csúcsokhoz rendelt aktivitás-valószínűségek állandósult állapotban vett eloszlásáról, és az Inverz Participációs Rátáról (IPR) mondhassunk valamit, szükségünk van még a $\underline{\underline{\boldsymbol{B}}}$ mátrix legnagyobb sajátértékére, valamint három legnagyobb sajátértékéhez tartozó sajátvektorára (a program jelenleg csak az elsőrendű tagok számítására képes). A sajátértékek számítása olyan lépés, amelyet szintén célszerű lehet a grafikus kártyával elvégeztetni, mivel számos lineáris művelet, elsősorban mátrix vektor szorzás elvégzését igényli.

Programom az imént áttekintett lépéseket végzi el. Kezdetben a CPU-hoz rendelt memóriaterületen előállítja a BAT modell algoritmusának lépésein keresztül az $\underline{\underline{\boldsymbol{A}}}$ mátrix alsó háromszög mátrixát, CSR formátumban, majd áthelyezi a grafikus kártya globális memóriájába. Két változat készült, az egyik parancssori argumentumként várja a kezdő hálózat elemszámát (a kódban: `int stN`), a hozzáadandó elemek számát (`int addN`), a csatlakozó élek számát (`int m`), a numerikus hibát

(`double eps`). A másik ugyan ezeket kéri be, de megadható az ismétlések száma is (`int rep`).

Miután a hálózat szomszédsági mátrixa a GPU globális memóriájába került, hatvány módszerrel állapítjuk meg $\underline{\underline{\boldsymbol{A}}}$ legnagyobb sajátértékét. Ha szükség van a másik kettőre is, akkor opcionálisan a Lánczos Kornél-féle tridiagonalizáció megelőzheti a hatvány módszer iterációit, ami jelentősen felgyorsíthatja az eljárást. A programhoz egy Lánczos-módszerrel működő modul elkészült ugyan, de ez nem került be a végleges változatba. Az IPR kalkuláció ideje jelentéktelennek bizonyult a CPU-n, még a GPU memóriáját majdnem teljesen kitöltő hálózatoknál is, ezért nem volt célszerű a grafikus feldolgozó egységhez rendelni ezt a műveletet.

3.2. A CSR ritka mátrix formátum

A CSR formátum használata mellet elsősorban azért döntöttem, mert a szakdolgozathoz készített programban, a sajátértékek számításához igyekeztem felhasználni azokat a lineáris algebrai rutinokat, amelyek a cuBLAS és cuSPARSE névre hallgató függvénykönyvtárakban elérhetőek. Mivel a cuSPARSE ritka mátrix szorzó algoritmusa ezt a formátumot támogatja, és a memória használatban jelentkező megtakarítás is így maximális, a CSR volt a legjobb választás.

Ahhoz, hogy a CSR formátum előnyeiről meggyőződhessünk, először egy régebbi, de még mindig népszerű ritka mátrix tárolási formával kell megismerkednünk, ami COO néven ismert. Egy mátrix COO formátumú tárolása során a nulla elemeket nem tároljuk, csak a nem nulla elemeket, valamint azok sor és oszlop indexeit [9]. Lesz tehát három tömbünk, egy a nem nulla elemeknek, egy a sor indexeknek és egy az oszlop indexeknek. Ennyi adat alapján az eredeti mátrix rekonstruálható.

A CSR formátum még tömörebb, ugyanis a sorok indexeinek teljes tömbjét nem jegyezzük meg. Helyette egy úgynevezett sor pointer tömböt alkotunk, amelyben azon elemek nem-nulla elem tömbben vett indexét helyezzük el, amelyeknél új sor kezdődik. Itt feltételeznünk kell, hogy a nem-nulla elemeket a megfelelő tömbbe sor-folytonosan olvastuk be. [10] Figyelni kell arra, hogy a leképezés a telt és a CSR forma között akkor is kölcsönösen egyértelmű maradjon, amikor a mátrix sorai között egy vagy több olyan akad, amelynek tisztán zérus elemei vannak. Ekkor a sor pointer tömbbe annak az elemnek a nem nulla elem tömbben vett indexét kell kétszer beírni, amely fölött a tisztán nullákból álló sor áll a telt mátrixban. Ha

egymás után több nulla sorról lenne szó, akkor még több ismétlés kell.

További megtakarítást jelenthet, ha a teljes mátrix helyett annak csak az alsó vagy felső háromszög mátrixát tároljuk. Megtehetjük, hiszen korábban rögzítettük, hogy a gráf nem rendelkezik irányított élekkel, így szomszédsági mátrixa is, és a $\underline{\underline{\boldsymbol{B}}}$ mátrix is szimmetrikus.

Bár a $\underline{\underline{\boldsymbol{B}}}$ mátrix tárolható ritka mátrixként, a vektor, amelyet a hatvány iteráció során újra-és-újra szorzunk vele, általában telt vektor, ahogy az eredményekre - a sajátvektorokra - sem igaz, hogy gyakoriak lennének bennük a nulla komponensek. Tehát a sajátvektort telt vektorként kezeltem.

3.3. Sajátérték számítás

Nagy mátrixok sajátértékeinek számítása leggyakrabban a QR dekompozíciós eljárással történik, amit szimmetrikus mátrixok esetén megelőzhet a Lánczos-féle tridiagonalizálás [11]. Ez az eljárás rendkívül hatékony, ugyanakkor az összes sajátértéket egyszerre szolgáltatja, és a sajátvektorok kinyeréséhez további transzformációkra van szükség. Számunkra csak a három legnagyobb sajátértékhez tartozó sajátvektor érdekes, sőt az esetek többségében csak a legnagyobb sajátérték sajátvektora határozza meg az állandósult állapot alakulását, ezért ebben a feladatban a QR dekompozíció nem hatékony megoldás. Célszerűnek látszik tehát az egyszerű hatvány iterációt használni a legnagyobb három sajátértékhez tartozó sajátvektor kereséséhez. Ennek a műveletnek az elvégzését érdemes a grafikus kártyára bízni, mert jól párhuzamosítható, és az eljáráshoz szükséges alapvető lineáris algebrai rutinok elérhetők a CUDA Developer Kit-el együtt szállított cuSPARSE és cuBLAS könyvtárakban.
Programomban a sajátérték számítást végző kódrészlet, mint a hatvány iteráció egy megvalósítása, a következő eljárást ismétli:

- Felveszünk egy egységnyi hosszúságú, véletlenszerű komponensekből álló vektort, legyen ez $\underline{\boldsymbol{q}}$. (A CPU-hoz tartozó memóriaterületen a tömb neve: `q`, a grafikus kártya globális memóriájában `d_q`.)
- A hálózat $\underline{\underline{\boldsymbol{A}}}$ mátrixával megszorozzuk a $\underline{\boldsymbol{q}}$ vektort, és az eredmény egy $\underline{\boldsymbol{r}}$ vektorba kerül.

- Mind $\underline{\boldsymbol{r}}$, mind $\underline{\boldsymbol{q}}$ normálása után az $\underline{\boldsymbol{r}}$-ből kivonjuk $\underline{\boldsymbol{q}}$-t, és a különbséget $\underline{\boldsymbol{q}}$ tömbjében helyezzük el.
- Megnézzük, hogy a különbségvektor normája kisebb e egy megfelelően választott ϵ számnál. (A programban `double eps` az ϵ változója.)
- Amennyiben nem, $\underline{\boldsymbol{r}}$ tartalmát $\underline{\boldsymbol{q}}$-ba másoljuk.
- Az eljárást addig ismételjük, míg a különbség kisebb nem lesz ϵ-nál.

Mikor a fenti ciklus véget ér, a $\underline{\boldsymbol{q}}$-val jelölt tömbben tárolt vektor megközelítőleg az $\underline{\underline{\boldsymbol{A}}}$ legnagyobb sajátértékéhez tartozó sajátvektora, de a sajátérték kiszámítása még hátra van. Legegyszerűbb, ha e célból a Rayleigh-kvócienst használjuk, ami:

$$\underline{\boldsymbol{q}}\left(\Lambda\right) \cdot \underline{\underline{\boldsymbol{A}}} \cdot \underline{\boldsymbol{q}}\left(\Lambda\right) = \Lambda \tag{3.1}$$

Tehát a program a ciklusból kilépve még végrehajtja a következő műveletsort:

- Megszorozza $\underline{\boldsymbol{r}}$-et $\underline{\underline{\boldsymbol{A}}}$-val, az eredmény $\underline{\boldsymbol{q}}$-ba kerül.
- Skalárisan összeszorozza $\underline{\boldsymbol{r}}$-et $\underline{\boldsymbol{q}}$-val.
- Visszaadja a skaláris szorzat eredményét, ami a sajátérték.

Miután a sajátérték ismert, még meg kell határozni az IPR értékét. Ezt már a CPU végzi el. Az IPR egyszerűen a legnagyobb sajátértékhez tartozó sajátvektor komponenseinek negyedik hatványa, amit a CPU-hoz tartozó dinamikus memóriaterületen lefoglalt q tömb elemein végigfutó for ciklus végez el.

Látható, hogy az analitikus számításokról szóló fejezetben ismertetett eljárás csak részben végezhető el a GPU erőforrásainak igénybevételével, aminek oka az, hogy a sajátérték számítással ellentétben a gráf előállítása, vagyis a gráfot jellemző szomszédsági mátrix elemeinek számítása nehezen illeszthető a grafikus kártyák masszívan párhuzamos architektúrájához. Bár vannak a Barabási-féle algoritmus párhuzamosítását célzó, viszonylag friss próbálkozások, ezek kissé eltérnek az eredeti algoritmustól a skálázhatóság kialakítása érdekében, ezért döntöttem a kettős kialakítás mellett [12].

Ha megfigyeljük a Barabási-Albert -féle, preferenciális csatolású fát generáló algoritmust, akkor láthatjuk, hogy abban az adott csúcsokhoz való csatlakozás valószínűségét minden lépésben újra kell számítani az alapján, hogy a korábbi lépésben

mely csúcshoz csatlakozott az új csúcs éle. Mivel ez a korábbi lépésekben kialakított konfigurációtól való függést jelent, nem végezhetjük el a generálás egyes lépéseit egymással párhuzamosan, azokat szigorúan az algoritmus által meghatározott sorrendben kell elvégeznünk, így a gráf generálás során fellépő műveletek közül csak egyes részfeladatok végezhetők a grafikus kártyán. Annak eldöntése, hogy az új csúcs melyik korábban csatlakozott csúcshoz kapcsolódik, a CPU feladata marad.

Két olyan részfeladat marad tehát, amelyek a GPU-ra delegálhatók. Egyik az élek súlyainak meghatározása, ha a szomszédságot kifejező $\underline{\underline{\boldsymbol{A}}}$ mátrix már ismert, a másik pedig az egyes iterációs lépések során a csatlakozás eldöntéséhez használt véletlen számok generálása. Előbbi feladat a WBAT-II súlyok esetére valósítható meg, egy saját CUDA kernel segítségével, utóbbit pedig a cuRAND függvénykönyvtár rutinjai végzik. Mivel a megfelelő minőségű véletlen számok generálása nagyon fontos feltétele egy korrekt szimuláció kidolgozásának, és a C standard véletlenszám generátora nem felelt meg elvárásainknak, a cuRAND rutinjait is meg kellett vizsgálni ebből a szempontból. Martin Weigel és szerzőtársai átfogó összehasonlító tesztet végeztek grafikus kártyás véletlen szám generátorokra, így a cuRAND függvényeire is. [13] Ebben a vizsgálatban a cuRAND függvények a szerzők saját fejlesztésű függvényeinél valamivel lassabbnak bizonyultak, azonban kiválóan teljesítettek a TestU1 könyvtár tesztjein, amely egy C könyvtár véletlen szám generátorok kvantitatív jellemzésére. Egyébként a cuRAND használatával a hálózat előállításához szükséges véletlen szekvencia a program teljes futásidejének töredéke alatt elkészül.

4. Eredmények, értékelés

Eddig megismerkedtünk a módszertani háttérrel, és a programmal, amellyel a számításokat elvégeztem, azonban hátra van még az eredmények ismertetése, és néhány következtetés, amit ezek kapcsán tehetünk. Minden méréssorozatban az IPR, és a sajátérték érdekelt elsősorban. A fokszámeloszlás különösen a kezdeti hálózat elemszámának a később kifejlődő hálózat fokszámeloszlásának alakulására gyakorolt hatása miatt érdekelt, szerettem volna a [14] ismert analitikus eredményeket a számítógépes szimuláció alapján kimutatni, és az eredményeket összevetni a hivatkozott cikkben látható ábrákkal.

4.1. Fokszámfüggés

A Barabási-Albert -féle hálózatot generáló algoritmus az eredeti cikkben [7] foglaltakon túl úgy is alkalmazható, hogy kezdetben megadunk egy tetszőleges hálózatot, amelynek mérete és fokszámeloszlása tetszőleges lehet. Az alapértelmezett esetben egy csúcsból indulunk, és a csatlakozó csúcsoknak a korábban bemutatott szabály szerint kell csatlakoznia a hálózathoz, azonban a kezdő hálózatot, az úgynevezett magot sokféleképp állíthatjuk elő, akár azt is megtehetjük, hogy egy másik gráf generáló algoritmus szerint építjük fel. Például lehet a hálózat magja egy Erdős-Rényi gráf, ahol a fokszámeloszlás $P(k) = \frac{t^k}{k!}e^{-kt}$ eloszlást követ, de lehetnek a kezdő elemek egy reguláris rács elemei, vagy az egyszerűség kedvéért összeköthetjük őket úgy, hogy minden elem legyen szomszédos minden más elemmel. Kérdéses, hogy ekkor vajon ugyanúgy $P(k) = \frac{1}{k^3}$ alakú lesz e aszimptotikusan a fokszámeloszlás, vagy a mag tulajdonságaitól, azaz fokszámeloszlásától és méretétől függően előfordulhat a hatványfüggvény szerű alaktól való eltérés.

Y. Berset, M. Medo a [14] cikkben megmutatta, hogy amennyiben a kezdeti feltétel olyan, hogy a magban az átlagos fokszám nagyobb háromnál, a Barabási -féle levezetésből ismert hatványfüggvény alakú fokszámeloszlás, azaz a korábban bemutatott

$$P(k) \sim \frac{1}{k^3} \tag{4.1}$$

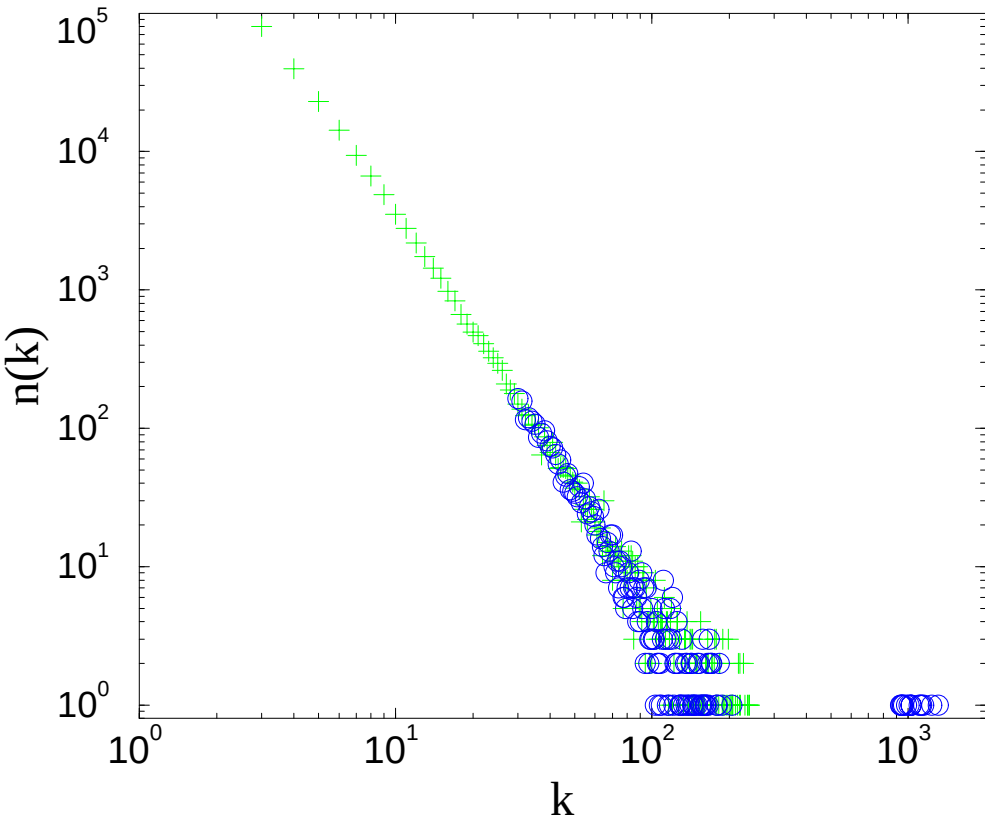
nem teljesül. Egész pontosan a fenti szerzőpáros egy olyan egyenlőtlenséget állított fel, amely abban az esetben érvényes, ha a kezdeti feltételként adott mag fokszámeloszlása normális eloszlású. Így az egyenlőtlenséget, amely feltételt ad arra, hogy mikor sérti meg a kezdeti feltétel a későbbi hálózat hatványfüggvény szerű fokszámeloszlásának kialakulását, olyan magok esetére írták fel, melyek topológiájáról nem esik említés, csupán azt feltételezik, hogy a fokszámeloszlás kezdetben:

$$P(k) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(k-\mu_0)^2}{2\sigma^2}} \tag{4.2}$$

Ekkor érvényes, hogy az adott magból kiindulva a felépített hálózat fokszámeloszlása akkor nem követi a hatványfüggvény szerű alakot, ha kezdetben az átlagos fokszámra (μ_0) érvényes: [14]

$$\mu_0 \geq 2\sqrt[3]{\pi} \approx 3 \tag{4.3}$$

Az általam vizsgált hálózatoknál mindig úgy indítottam a generáló algoritmust, hogy a kezdeti csomópontok teljesen összekötött rendszert alkottak, mindegyik csúcs a hálózat gráfjában szomszédja volt az összes többinek. Azt tapasztaltam, hogy a fenti szabály ekkor is érvényes, különféle méreteknél. Saját kezdőfeltételemmel elvégezve a hálózat generálását az eredmény igazolta 4.3-et. $N = 200000$, $m = 3$, és $N_0 = 3$, valamint $N = 200000$, $m = 3$, $N_0 = 30$ esetre a fokszámeloszlásokat log-log skálán ábrázoltam. A fokszámeloszlás log-log skálán ábrázolva láthatóan követi az $N_0 = 3$ esetre az analitikus számítások alapján várható alakot:



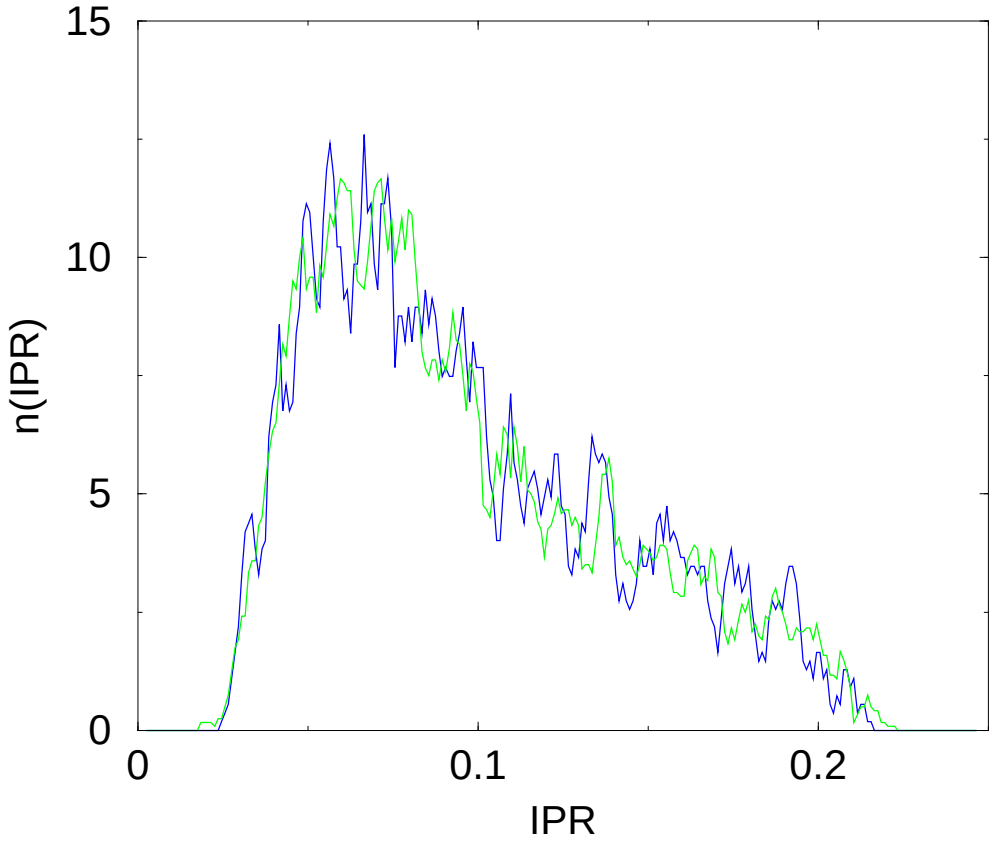
4.1. ábra: Fokszámok gyakorisága, kék az $N_0 = 30$-as eset, a zöld $N_0 = 3$

Azonban, ha a kezdő elemszám elég nagy (itt $N_0 = 30$), a nagy fokszámú kezdő csúcsok leválnak a hatványfüggvény szerű eloszlásról, és kiugróan sok élt gyűjtenek.

Azért ábrázoltam $N = 200000$ méretű hálózatot, mert nagyjából ekkor kezd a jelenség megfigyelhetővé válni, nagyobb hálózatoknál látványosabb az effektus. A kezdeti feltétellel tehát óvatosan kell bánni, mert érdemben befolyásolhatja a topológiát, ami viszont QMF közelítésben a kritikus λ és az IPR alakulását megváltoztathatja.

4.2. Hiba vizsgálata

A programban a hatvány módszernél az elvárt konvergencia kritérium (ϵ) parancssori argumentumként megadható. Ennek kezdetben úgy véltem lehet némi szerepe abban, hogy az IPR értékében jelentős fluktuációt tapasztaltam. Többféle méretű hálózatra elvégeztem egy olyan mérés-sorozatot, ahol ugyanarra a méretre vizsgáltam az IPR és a legnagyobb sajátérték alakulását, kétféle numerikus pontosság mellett ($\epsilon = 0.001$ és 0.00001). Ezek ezer fölötti ismétlésszámmal készültek a Szegedi Tudományegyetem grafikus kártyás számítógép klaszterén, és az eredmények megtalálhatóak a szakdolgozathoz mellékelt lemezen. Az alábbi ábrán $N = 50000$, $N_0 = 3$, $m = 3$-nál látható az IPR, 3000 elemű minta alapján:



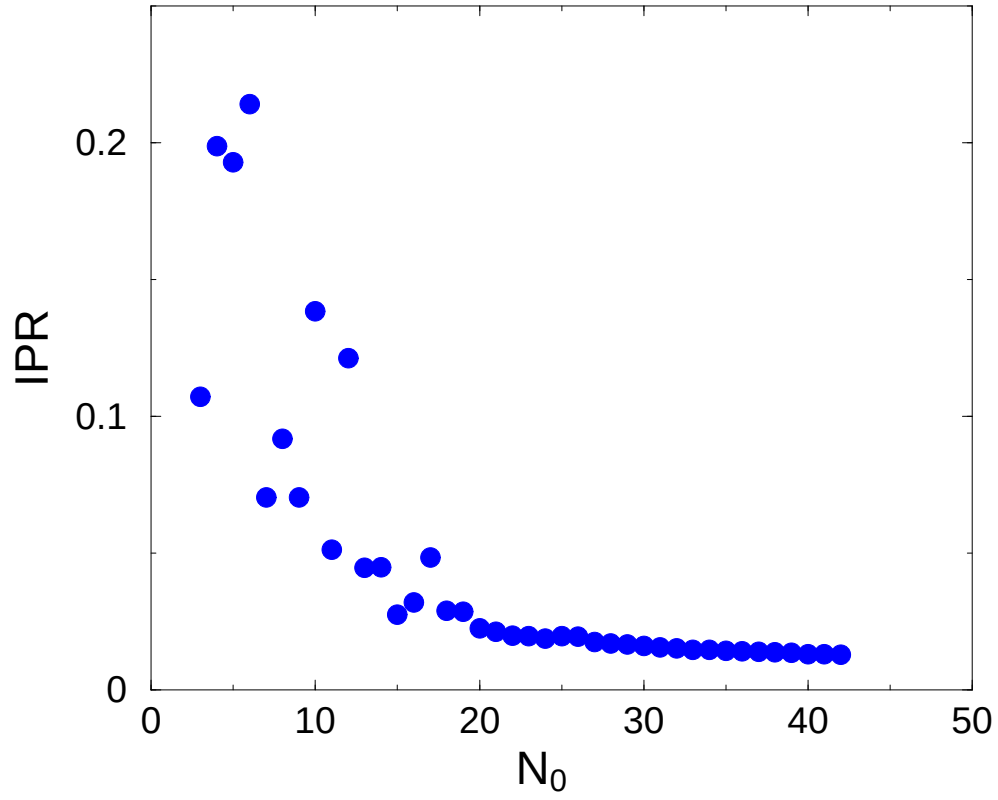
4.2. ábra: IPR tapasztalati valószínűség sűrűsége, különböző ϵ esetén, zöld: $\epsilon = 0.001$, kék: $\epsilon = 0.00001$

Látható, hogy a két adatsor erősen korrelált, a keresztkorrelációs együttható nagyjából 0.97, ami azt jelenti, hogy a magas szórás elsősorban nem a numerikus

hiba miatt jellemzi a rendszert. A program e tekintetben, és a fokszámeloszlások vizsgálata során is megfelelően pontosnak bizonyult, így alkalmas arra, hogy vele részletesebb vizsgálatokat lehessen végezni, ami fényt deríthet arra, hogy vajon miért olyan magas az IPR szóródása. Ehhez többszöri futtatásra, a teszteléshez használtaknál nagyobb mintaméretekre lesz szükség.

4.3. IPR, a kezdeti feltétel függvényében

Miután ellenőriztem a sajátérték számításért felelős kódrészlet működésének helyességét, és a hálózatot generáló algoritmust, készítettem egy mérést $N = 200000$ méretű, $m = 3$-as, BA hálózatokról úgy, hogy a kezdő mag méretét egyesével növeltem $N_0 = 3$-tól 40 lépésen keresztül:



4.3. ábra: IPR érték, különböző N_0 esetén, egy-egy realizációra

Mivel a hálózat generálásának indításakor meghatározott hálózatban az átlagos fokszám értéke hatással lehet a hálózat későbbi fokszámeloszlására, és a maximális fokszámra, a [15]-ben közölt $\lambda = \sqrt{k_{max}}$ összefüggés miatt a legnagyobb sajátérték,

ezzel együtt esetleg az IPR is másképp viselkedhet. Egy olyan hálózat, amelyben a kezdeti csúcsok mindannyian össze vannak kötve, és háromnál többen vannak, egyértelműen kívül esik azon kezdeti feltételek körén, amelyekből kiindulva a fokszámeloszlás meg fog felelni az eredeti Barabási-Albert cikkben felvázolt hatványfüggvény szerű alaknak. Nem meglepő tehát, hogy az egyszerű, súlyozatlan Barabási-Albert gráfnál az IPR kimondottan érzékenynek látszik a kezdeti hálózat átlagos fokszámeloszlására, főleg $N_0 = 20$ fölött, ami intuíciónkkal nagyjából összhangban is van. Ezek a mérések azonban kizárólag a súlyozatlan esetre vonatkoztak, nem érintettük a WBAT-I, WBAT-II, B-fa hálózatokat.

5. Összegzés

Szakdolgozatom kezdeti célkitűzései közül sikerült teljesíteni a legfontosabbat, hogy egy gyorsabb eszközt bocsássak témavezetőm rendelkezésére a nem-egyensúlyi statisztikus fizikai rendszerek kutatásához. A tesztelés során kapott eredményekből az derült ki, hogy számos esetben a működés hibátlan, azonban előfordul, hogy - főleg fa szerkezetű hálózatoknál - az egyszerű hatvány módszerrel nem lehet elérni kellő numerikus hibahatáron belül eső konvergenciát, ezért a program továbbfejlesztésének legfontosabb iránya az lehetne, hogy egy másik sajátérték megoldó algoritmust használjon. Sokat lehetne javítani a futásidőn, ha a [10]-ben leírt mátrix-szorzó CUDA kernelt, és a hozzá optimalizált ritka-mátrix formátumot használnánk. De jelentős minőségi előrelépés lenne egy saját kernel készítése, amellyel a WBAT-II hálózatokon az élek súlyait a GPU feladata lenne bevezetni. Ekkor a programmal lehetne súlyozott hálózatokat vizsgálni, ami fontos követelmény ezen a kutatási területen.

A SIS modell kapcsán sikerült kimutatni azt, hogy a kezdeti hálózat átlagos fokszámaitól függhet a BA hálózatok aszimptotikus fokszámeloszlása, ami hatással lehet az IPR és a sajátérték, így a kritikus lambda állandósult állapotban vett értékére. Ezen állítások megfelelő alátámasztására azonban még nem elegendő a generált adatmennyiség, így messzemenő következtetések levonására nincs lehetőség. A jövőben tehát célszerű a program hiányzó képességeinek megvalósítása után egy szisztematikus mérés-sorozatot végezni, amelyben megállapítható az IPR szóródásának fokszámtól és a kezdő halmaz elemszámától való függése, hogy az eredményekhez robusztus mintákból lehessen várható értéket és hibát számolni.

6. A program használata

Amennyiben érdeklik az olvasót az eddigi futások eredményei, illetve a program maga, a szakdolgozathoz mellékelt CD-n mind az adatsorokat mind a program forráskódját megtalálja. A forráskód fordításához az nVIDIA CUDA Developer kit és egy friss grafikus kártya illesztőprogram letöltése szükséges. Linux rendszeren a fordítás a következő utasítással történik:

```
-nvcc -lcublas -lcusparse -lcurand forraskod.cu -o nweig
```

Előfordulhat hogy a program 1.3-as vagy régebbi architektúrájú grafikus kártyákon lassabban fut, mert ezek a régebbi szériák nem támogatták a dupla lebegő pontos aritmetikát, ezért a program működése 2.0-ás architektúrájú és újabb grafikus kártyákon optimális.

Lehet, hogy a számítógépben rendelkezésre álló 2.0 architektúrájú grafikus kártya, azonban a fordító alapértelmezetten nem erre fordít. Ekkor használjuk a `-arch=compute_20` kapcsolót.

Ha a futtatható állomány előállt, egyszerű parancssoros alkalmazásként lehet futtatni. A beállítható argumentumok rendre: fájl neve, a kezdeti csúcsokhoz adandó elemek száma, a kezdő csúcsok száma, csatlakozó élek száma, numerikus pontosság.

```
./programnév kimenet.kiterjesztés (N-N_0) N_0 m eps
```

Például nézzük, hogyan indíthatnánk a programot egy 200 000 elemű hálózatra, amely egy 3 csúcsot számláló magból úgy növekszik, hogy a csatlakozó csúcsok minden körben 3 élt létesítenek a korábbi elemek közül hárommal, és a sajátérték számítás numerikus hibáját jellemző $\epsilon = 0.00001$:

```
./nweig adatok.dat 199997 3 3 0.00001
```

A használat tehát egyszerű, és megfelelő script segítségével a többszöri, változó argumentumokkal történő futtatásra is fel van készítve a program. Amennyiben script nélkül szeretnénk ugyanolyan paraméterekkel több realizációt előállítani, a program `nweigB` névre hallgató változatát kell használni. Itt a parancssori argumentumok a következők:

```
./programnév kimenet.kiterjesztés (N-N_0) N_0 m ismétlésszám eps
```

Zárásként megjegyzem, hogy a függelékben az `nweig` változat látható, az `nweigB` forráskódja pedig felkerült a szakdolgozathoz mellékelt CD-re.

Melléklet

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cuda_runtime.h>
#include "device_launch_parameters.h"
#include <cublas_v2.h>
#include <cusparse_v2.h>
#include <curand.h>

int PrintError(cudaError_t ERROR)
{
  bool kapcsolo=0;
  switch(ERROR)
  {
    case cudaSuccess:
    break;

    case cudaErrorMissingConfiguration:
    std::cout<<"cudaErrorMissingConfiguration"<<std::endl;kapcsolo=1;
    break;

    case cudaErrorMemoryAllocation:
    std::cout<<"cudaErrorMemoryAllocation"<<std::endl;kapcsolo=1;
    break;

    case cudaErrorInitializationError:
    std::cout<<"cudaErrorInitializationError"<<std::endl;kapcsolo=1;
    break;

    case cudaErrorLaunchFailure:
    std::cout<<"cudaErrorLaunchFailure"<<std::endl;kapcsolo=1;
    break;

    case cudaErrorPriorLaunchFailure:
    std::cout<<"cudaErrorPriorLaunchFailure"<<std::endl;kapcsolo=1;
    break;

    case cudaErrorInvalidValue:
    std::cout<<"cudaErrorInvalidValue"<<std::endl;
    break;

    case cudaErrorInvalidSymbol:
    std::cout<<"cudaErrorInvalidSymbol"<<std::endl;
    break;

    case cudaErrorInvalidDevicePointer:
    std::cout<<"cudaErrorInvalidDevicePointer"<<std::endl;
    break;
```

```
    case cudaErrorInvalidMemcpyDirection:
    std::cout<<"cudaErrorInvalidMemcpyDirection"<<std::endl;
    break;

    default:
    std::cout<<"error not specified"<<std::endl;kapcsolo=1;
    break;
  }
  return kapcsolo;
}

int CuspPrintStatus(cusparseStatus_t status)
{
  bool kapcsolo=0;
  switch(status)
  {
    case CUSPARSE_STATUS_SUCCESS:
    break;

    case  CUSPARSE_STATUS_NOT_INITIALIZED:
    std::cout<<"CUSPARSE_STATUS_NOT_INITIALIZED"<<std::endl;kapcsolo=1;
    break;

    case  CUSPARSE_STATUS_ALLOC_FAILED:
    std::cout<<"CUSPARSE_STATUS_ALLOC_FAILED"<<std::endl;kapcsolo=1;
    break;

    case   CUSPARSE_STATUS_INVALID_VALUE:
    std::cout<<"CUSPARSE_STATUS_INVALID_VALUE"<<std::endl;kapcsolo=1;
    break;

    case   CUSPARSE_STATUS_ARCH_MISMATCH:
    std::cout<<"CUSPARSE_STATUS_ARCH_MISMATCH"<<std::endl;kapcsolo=1;
    break;

    case    CUSPARSE_STATUS_EXECUTION_FAILED:
    std::cout<<"CUSPARSE_STATUS_EXECUTION_FAILED"<<std::endl;kapcsolo=1;
    break;

    case    CUSPARSE_STATUS_MAPPING_ERROR:
    std::cout<<"CUSPARSE_STATUS_MAPPING_ERROR"<<std::endl;kapcsolo=1;
    break;

    case    CUSPARSE_STATUS_INTERNAL_ERROR:
    std::cout<<"CUSPARSE_STATUS_INTERNAL_ERROR"<<std::endl;kapcsolo=1;
    break;

    case    CUSPARSE_STATUS_MATRIX_TYPE_NOT_SUPPORTED:
    std::cout<<"CUSPARSE_STATUS_MATRIX_TYPE_NOT_SUPPORTED"<<std::endl;kapcsolo=1;
    break;
```

```
    default:
    std::cout<<"error not specified"<<std::endl;kapcsolo=1;
    break;
  }
  return kapcsolo;
}

int main(int argc, char** argv)
{
  std::string filenev=argv[1];
  std::ofstream kimenet(filenev.c_str());

  clock_t begin=0; clock_t end=0; double tm=0;

  begin=clock();

  int addN=atoi(argv[2]);
  int stN=atoi(argv[3]);
  int m=atoi(argv[4]);

  int N=addN+stN; int limit_rw=N+1;
  int M=stN*(stN-1)/2+m*addN; int limit_nz=M;

  double* deg=(double*)malloc(N*sizeof(double));
  for(int i=0;i<stN;i++)
  {
    deg[i]=stN-1;
  }

  double* mtxCSRnz=(double*)malloc((limit_nz)*sizeof(double));if(mtxCSRnz==0){return 1;}
  int* mtxCSRcl=(int*)malloc((limit_nz)*sizeof(int));if(mtxCSRcl==0){return 1;}
  int* mtxCSRrw=(int*)malloc((limit_rw)*sizeof(int)); mtxCSRrw[0]=0;mtxCSRrw[1]=0;
  if(mtxCSRrw==0){return 1;}

  for(int i=2;i<stN;i++)
  {
    mtxCSRrw[i]=i-1+mtxCSRrw[i-1];
}
  mtxCSRrw[stN]=stN-1+mtxCSRrw[stN-1];
  for(int i=stN+1;i<N;i++)
  {
    mtxCSRrw[i]=m+mtxCSRrw[i-1];
}
  mtxCSRrw[N]=limit_nz;

  int index=0;
  for(int i=1;i<stN;i++)
  {
    for(int j=0;j<i;j++)
    {
```

```
    mtxCSRnz[index+j]=1;
    mtxCSRcl[index+j]=j;
  }
  index+=i;
}

double rnd=0;

cudaDeviceProp devprop;
int device=0;
cudaGetDevice(&device);cudaGetDeviceProperties(&devprop,device);
cudaSetDevice(0);
cudaGetDevice(&device);cudaGetDeviceProperties(&devprop,device);

cublasHandle_t blaskez;
cublasCreate(&blaskez);
cusparseHandle_t sparsekez;
cusparseCreate(&sparsekez);
cusparseMatDescr_t leir;
cusparseCreateMatDescr(&leir);

cusparseStatus_t status;

status=cusparseSetMatIndexBase(leir,CUSPARSE_INDEX_BASE_ZERO);CuspPrintStatus(status);
status=cusparseSetMatType(leir,CUSPARSE_MATRIX_TYPE_SYMMETRIC);CuspPrintStatus(status);
status=cusparseSetMatFillMode(leir,CUSPARSE_FILL_MODE_LOWER);CuspPrintStatus(status);
status=cusparseSetMatDiagType(leir,CUSPARSE_DIAG_TYPE_NON_UNIT);CuspPrintStatus(status);

curandGenerator_t generator;
curandCreateGeneratorHost(&generator,CURAND_RNG_PSEUDO_MTGP32);
curandSetPseudoRandomGeneratorSeed(generator,(unsigned long)time(0));



double* random=(double*)malloc(m*(N-stN)*sizeof(double)); if(random==0){return 1;}
curandGenerateUniformDouble(generator,random,m*(N-stN));

bool kihagyas=0;
int csere=0; int idg=0;
double sumdeg_maradek=0;

int tiltott_indexek[m]; int idg_cl[m];

for(int i=0;i<m;i++)
{
  idg_cl[m]=0;
}

for(int i=stN,sumdeg=stN*(stN-1);i<N;i++)
{
  sumdeg_maradek=sumdeg;
```

```
for(int f=0;f<m;f++)
{
  tiltott_indexek[f]=-1;
}

for(int l=0;l<m;l++)
{
  rnd=random[(i-stN)*m+l];

  for(int j=0;j<i;j++)
  {

    for(int h=0;h<m;h++)
    {
      if(tiltott_indexek[h]==j)
      {
        kihagyas=1;
        break;
      }
    }

    if(kihagyas==0)
    {
      if(rnd<deg[j]/sumdeg_maradek)
      {
        sumdeg_maradek-=deg[j];
        deg[j]++;
        deg[i]++;
        mtxCSRnz[index+(i-stN)*m+l]=1;
        idg_cl[l]=j;
        tiltott_indexek[l]=j;
        sumdeg+=2;
        break;

      }
      else if(j==(i-1))
      {
        sumdeg_maradek-=deg[j];
        deg[j]++;
        deg[i]++;
        mtxCSRnz[index+(i-stN)*m+l]=1;
        idg_cl[l]=j;
        tiltott_indexek[l]=j;
        sumdeg+=2;
        break;
      }
      else
      {
        rnd=rnd-deg[j]/sumdeg_maradek;
      }
```

```
      }
      else
      {
        kihagyas=0;
      }
    }
  }

  do
  {
    csere=0;
    for(int l=0;l<m-1;l++)
    {
      if(idg_cl[l]>idg_cl[l+1])
      {
        idg=idg_cl[l+1];
        idg_cl[l+1]=idg_cl[l];
        idg_cl[l]=idg;
        csere++;
      }
    }
  }
  while(csere!=0);

  for(int l=0;l<m;l++)
  {
    mtxCSRcl[index+(i-stN)*m+l]=idg_cl[l];
  }
}

free(random);

for(int i=0;i<N;i++)
{
  kimenet<<deg[i]<<std::endl;
}

cudaError_t error;
double* d_mtxCSRnz=0; error=cudaMalloc(&d_mtxCSRnz,(limit_nz)*sizeof(double));
if(PrintError(error)==1)return 1;
int* d_mtxCSRcl=0; error=cudaMalloc(&d_mtxCSRcl,(limit_nz)*sizeof(int));
if(PrintError(error)==1)return 1;
int* d_mtxCSRrw=0; error=cudaMalloc(&d_mtxCSRrw,(limit_rw)*sizeof(int));
if(PrintError(error)==1)return 1;

error=cudaMemcpy(d_mtxCSRnz,mtxCSRnz,(limit_nz)*sizeof(double),cudaMemcpyHostToDevice);
error=cudaMemcpy(d_mtxCSRcl,mtxCSRcl,(limit_nz)*sizeof(int),cudaMemcpyHostToDevice);
error=cudaMemcpy(d_mtxCSRrw,mtxCSRrw,(limit_rw)*sizeof(int),cudaMemcpyHostToDevice);

free(mtxCSRnz); mtxCSRnz=0; if(PrintError(error)==1)return 1;
free(mtxCSRrw); mtxCSRrw=0; if(PrintError(error)==1)return 1;
```

```
free(mtxCSRcl); mtxCSRcl=0; if(PrintError(error)==1)return 1;

double* q=(double*)malloc(N*sizeof(double)); if(q==0){return 1;} for(int i=0;i<N;i++){q[i]=1;}
double* d_q=0; error=cudaMalloc(&d_q,N*sizeof(double));if(PrintError(error)==1)return 1;
double* d_r=0; error=cudaMalloc(&d_r,N*sizeof(double));if(PrintError(error)==1)return 1;
error=cudaMemcpy(d_q,q,N*sizeof(double),cudaMemcpyHostToDevice);


double nulla=0;double egy=1;
double alfa=0;double eps=double(atof(argv[5]));

int nem_konvergal=0;

do
{
  status=cusparseDcsrm(sparsekez,CUSPARSE_OPERATION_NON_TRANSPOSE,N,N,limit_nz,&egy,
  leir,d_mtxCSRnz,d_mtxCSRrw,d_mtxCSRcl,d_q,&nulla,d_r);
  cublasDnrm2(blaskez,N,d_r,1,&alfa);
  alfa=1/alfa;
  cublasDscal(blaskez,N,&alfa,d_r,1);
  alfa=-1;
  cublasDaxpy(blaskez,N,&alfa,d_r,1,d_q,1);
  cublasDnrm2(blaskez,N,d_q,1,&alfa);
  cublasDcopy(blaskez,N,d_r,1,d_q,1);
  nem_konvergal++;
}
while(alfa>=eps&&nem_konvergal<100000);

cublasDnrm2(blaskez,N,d_q,1,&alfa);
alfa=1/(alfa);
cublasDscal(blaskez,N,&alfa,d_q,1);

status=cusparseDcsrmv(sparsekez,CUSPARSE_OPERATION_NON_TRANSPOSE,N,N,limit_nz,&egy,
leir,d_mtxCSRnz,d_mtxCSRrw,d_mtxCSRcl,d_q,&nulla,d_r);
CuspPrintStatus(status);
cublasDdot(blaskez,N,d_q,1,d_r,1,&alfa);

error=cudaMemcpy(q,d_q,N*sizeof(double),cudaMemcpyDeviceToHost);
if(PrintError(error)==1)return 1;

double IPR=0;
for(int i=0;i<N;i++)
{
  IPR=IPR+q[i]*q[i]*q[i]*q[i];
}

end=clock(); tm=(double)(end-begin)/CLOCKS_PER_SEC;

std::cout<<IPR<<'\t'<<alfa<<std::endl;

cudaFree(d_mtxCSRnz); d_mtxCSRnz=0;
```

```
  cudaFree(d_mtxCSRcl); d_mtxCSRcl=0;
  cudaFree(d_mtxCSRrw); d_mtxCSRrw=0;

  cudaFree(d_q); d_q=0;
  cudaFree(d_r); d_r=0;

  free(q); q=0; free(deg); deg=0;


  curandDestroyGenerator(generator);

  cublasDestroy(blaskez);
  cusparseDestroyMatDescr(leir);
  cusparseDestroy(sparsekez);

  kimenet.close();

  return 0;
}
```

Hivatkozások

[1] Ginestra Bianconi. *Superconductor-insulator transition in a network of 2d percolation clusters*. EPL, 101 26003, 2013

[2] Marian Boguna; Claudio Castellano; Romualdo Pastor-Satorras. *The nature of epidemic thresholds in networks*. arXiv:1305.4819, 2013

[3] Hyun Keun Lee; Pyoung-Seop Shim and Jae Dong Noh. *Epidemic threshold of susceptible-infected-susceptible model on complex networks*. arXiv:1211.2519, 2012

[4] Ódor Géza. *Rare regions of the sis model on barabási-albert networks*. Phys. Rev. E 87; 042132, 2013

[5] S. N. Dorogovtsev. *Critical phenomena in complex networks*. Rev. Mod. Phys. 80; 1275, 2008

[6] Ulrik Brandes; Thomas Erlebach (Eds.). *Network analysis methodological foundations*. Springer, 1998

[7] Réka Albert and Albert-László Barabási. *Statistical mechanics of complex networks*. Rev. Mod. Phys. 74, pages 47–97, 2002

[8] Fan Chung; Linyuan Lu. *Complex graphs and networks*. American Mathematical Soc., 2006

[9] nVIDIA, 2012. CuSPARSE CUDA Toolkit Documentation v5.0 http://docs.nvidia.com/cuda/pdf/CUDA_CUSPARSE_Users_Guide.pdf utolsó letöltés: 2013.05.30.

[10] Tomas Oberhuber; Atsushi Suzuki; Jan Vagata. *New row-grouped csr format for storing sparse matrices on gpu with implementation in cuda*. Acta Technica 56, pages 447–466, 2011

[11] Owe Axelsson. *Iterative Methods for Large Linear Systems*. North Holland, 1994

[12] Andy Yoo; Keith Henderson. *Parallel generation of massive scale-free graphs*. arXiv:1003.3684, 2010

[13] Markus Manssen; Martin Weigel and Alexander K. Hartmann. *Random number generators for massively parallel simulations on gpu*. Eur. Phys. J. Special Topics 210, 53:151–159, 2013

[14] Yves Berset and Matus Medo. *The effect of the initial network configuration on preferential attachment*. arXiv:1305.0205, 2013

[15] Arda Halu; Silvano Garnerone; Alessandro Vezzani; Ginestra Bianconi. *Phase transition of light on complex quantum networks*. Physical Review E 87; 022104, 2013

NYILATKOZAT

Név: Domokos Zoltán

ELTE Természettudományi Kar, szak: Fizika BSc

NEPTUN azonosító: VYL2MH

Szakdolgozat címe:
Nemegyensúlyi rendszerek vizsgálata grafikus kártyás szuperszámítógéppel

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló munkám eredménye, saját szellemi termékem, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2013. május 31.

_____

a hallgató aláírása