GPU-n futtatható neutronszimulációs kód fejlesztése

TDK dolgozat

Takács Hajna
Fizikus MSc hallgató
BME TTK

Témavezető: Dr. Légrády Dávid
egyetemi docens
BME Nukleáris Technikai Intézet,
Nukleáris Technika Tanszék

Budapesti Műszaki és Gazdaságtudományi Egyetem
2015

Kivonat

Monte Carlo (MC) alapú szimulációkat a tudományos kutatások és az ipar számos területén használnak, annak ellenére, hogy rendkívül erőforrásigényesek, cserébe a fizikai modellezés pontos, szemben a determinisztikus számításokkal, amikor nagyobb közelítésekkel, de gyorsabban számolhatunk. A megfelelően kicsi statisztikus szórás eléréséhez azonban nagy mintaszám szükséges, amellyel együtt nő a felhasznált gépidő és annak költsége. Tranziensek vizsgálatakor ez a megállapítás hatványozottan igaz.

Az atomreaktorban az egyes indított neutronok függetlenek egymástól, így elméletben a részecsketranszport kód könnyedén párhuzamosítható, amely lehetőséget teremt a grafikus számítási egységeken (GPU-kon) való hatékony működésre, alacsony bekerülési költség mellett. Ehhez azonban speciálisan erre az eszközre kifejlesztett algoritmusra van szükség, amely megalkotása és optimalizálása új kihívások elé állítja a fejlesztőt, hiszen a számítási architektúra jelentősen eltér a megszokott CPU- alapútól.

Munkám során egy olyan GPU Monte Carlo neutronszimulációs kód fejlesztésébe fogtam, amely alkalmas lehet reaktordinamikai tranziensek vizsgálatára. Ennek eléréséig még hosszú út áll a feladaton dolgozó csapat előtt, ezért jelenlegi célom annak feltérképezése, hogy mely algoritmusok lehetnek megfelelőek és melyek azok, amelyekkel biztosan nem érdemes tovább foglalkozni. Ezért kezdetben a lehető legegyszerűbb geometriai és anyagi modellt igyekeztem használni, amely már az összes fontosabb reaktordinamikai szempontból fontos kölcsönhatást modellezi.

Korábbi eredmények rendelkezésre állnak egy egyszerű téglatest modell szimulációjára [1] [2], azonban ezen algoritmusok egyáltalán nem GPU-kompatibilisek. Ennek ellenére a reaktorfizikai események és korlátok tekintetében hasznos kiindulási alapként szolgáltak.

Az algoritmusverziók vizsgálata során kiderült, hogy a valóság lemásolására törekvő analóg technikákkal a szimuláció még egyszerűbb esetben sem konvergens és a konvencionális egyszerűbb szóráscsökkentő eljárások sem segítenek. A két konvergáló algoritmusverzió fejlesztésével és optimalizálásával vontam le a továbblépéshez szükséges tanulságokat.

Tartalomjegyzék

1. fejezet

Bevezetés

A reaktorok biztonságos tervezésének és szabályozásának megalapozását jelentő jelenlegi számítások figyelemreméltó alternetívája a grafikus kártyák használata, amely olcsó és gyors számítási kapacitást biztosít a pontos reaktordinamikai szimulációkhoz, akár dinamikus visszacsatolással.

Reaktorfizikai számítások elvégzésére számtalan módszert fejlesztettek ki az utóbbi néhány évtizedben. A determinisztikus módszerek valamilyen fizikai, folyamatmodellezési vagy diszkretizációs közelítést alkalmaznak, amely sok esetben megfelelő üzemeltetési szempontból, azonban a jelenlegi célunk a reaktordinamikai jelenségek vizsgálata, ahol a pontosság iránti igény fokozottabb.

A Monte Carlo szimulációk a részecskék (jelen esetben neutronok) életútjának követésével, a reakciók megfelelő valószínűségi eloszlások segítségével való sorsolásának módszerével, valamint egy sor szóráscsökkentő, hatékonyságnövelő módszer használatával alternatívát jelentenek a determinisztikus módszerekkel szemben. A sztochasztikus módszerek hátránya a statisztikus szórás, amely sok esetben csak nagy mintaszámmal csökkenthető. Ennek a statisztikus szórás szempontjából megfelelően nagy mintaszámnak a növelésével azonban a futásidők is hamar megnőnek, akár kezelhetetlen mértéket öltenek. A nagy futási idő egyik oka az, hogy a részecskék életútját a számítógép processzora egymás után számítja.

Mivel az egyes részecskék életútja független egymástól, a szimuláció jól párhuzamosítható. Bizonyos körülmények között a CPU (Central Processing Unit – központi feldolgozóegység) támogatja az adatpárhuzamosságot, így az algoritmus nagyobb mérvű változtatása nélkül használhatunk egyszerre több magot, de ez jelentősen megdrágítja a munkát, hiszen lényegében több processzor kapacitását foglaljuk le, melyek beszerzése és üzemeltetése költséges.

A statisztikus szórás mértéke már statikus esetben is aggasztó lehet kellő részletességű, realisztikus modell esetén, tranziensek meghatározásakor pedig különösen nagy szerepet játszik, hiszen tapasztalataink szerint a gyors változásokhoz nagy szórás fog társulni. Éppen ezért tranziensek szimulálására kevés kísérlet történt, azok

is a végletekig egyszerűsítve [1]. Erre a problémára kínálnak megoldási lehetőséget a grafikus számítási egységek, amelyek képesek töredék bekerülési költség mellett a feladatok nagyszámú párhuzamosítására.

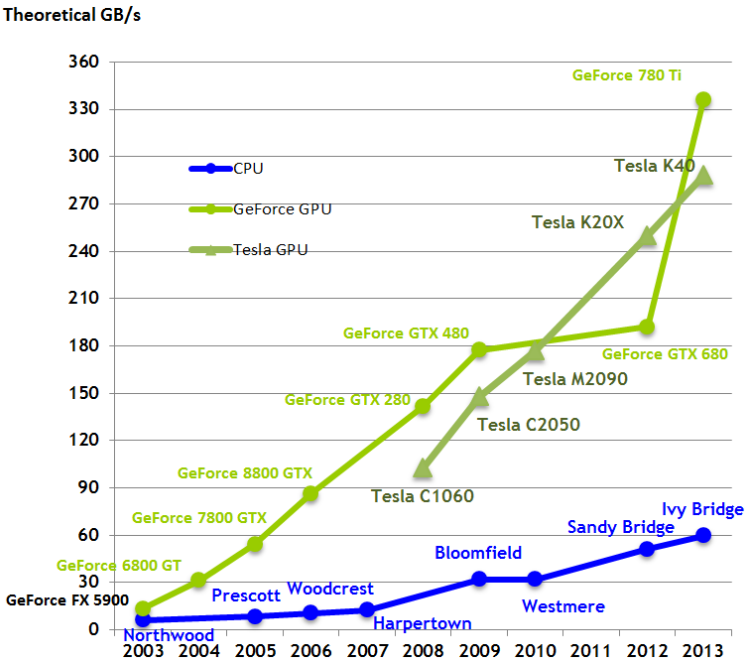
1.1. GPU felépítése

A GPU (Graphics Processing Unit) a grafikus vezérlőkártya központi egysége, amely általában az összetett grafikus műveletek elvégzéséért felelős. A GPU feladata a magas szintű feladatok átvétele a CPU-tól, annak tehermentesítésére, hiszen kizárólag az utóbbi alkalmas a programok vezérlésére, a gyorsabb utasítás kiválasztásra.

1.1. ábra. CPU és GPU felépítése közötti különbségek [3]

A GPU-k használata a megszokottól kissé eltérő programozást kíván, hiszen felépítésük jelentősen eltér a CPU-hoz képest. Fő különbségek az 1.1 ábrán figyelhetők meg. A GPU és a hozzá kapcsolódó dedikált memória közti kommunikáció lényegesen gyorsabb, mint a CPU és a rendszermemória közötti, mivel számítási műveleteket az úgynevezett multiprocesszorok (MP) végzik, melyek mindegyike önálló regiszterekkel és osztott memóriával rendelkezik. A memória sávszélességek jelentős különbségét és ennek időbeli fokozódását jól mutatja az 1.2 ábra. Továbbá a GPU aritmetikai logikai egységekkel (ALU) jobban fel van szerelve, így gyorsabban számol, de ennek ára, hogy az MP-ken warpok vannak definiálva, melyek egyszerre ugyanazt a műveletet végzik, különböző adatokon, akkor is, ha erre nincs szükség. Érdemes tehát olyan algoritmust készíteni, amely minél több felesleges munkát spórol meg az eszköznek. A GPU szempontjából hatékony algoritmus ismérve, hogy kevés vezérlő utasítást tartalmaz, azonban nagy számításigényű.

Annak ellenére, hogy a GPU-t elsősorban a grafikus megjelenítés eszközének szánták, hamar kiderült, hogy egyéb számításigényes feladatok elvégzésére is jól használható. Ilyen például a dolgozatomban tárgyalt Monte Carlo alapú neutronszimuláció is. A munkát megfelelő saját grafikus kártya hiányában a BME NTI AWING nevű szerverén végeztem, ahol két GeForce GTX 690 állt a rendelkezésemre 4 szállal, Linux környzetben, távoli eléréssel.



1.2. ábra. CPU-GPU memória sávszélessége [3]

1.2. CUDA programnyelv

A kódot az Nvidia saját grafikus kártyáinak programozására szolgáló CUDA programnyelven írtam, amely a C++ nyelven alapul néhány speciális utasítással. GPU programozásra a jelenleg rendelkezésre álló lehetőségek közül ez a legsokoldalúbb, azonban a manapság CPU programozásra használt nyelvekhez képest korlátozott eszköztárral rendelkezik. Alapvetően egy egyszerű C++ kódról van szó, melyen belül a GPU által futtatott függvényeket, úgynevezett kerneleket indíthatunk. A megfelelően lefoglalt memóriába töltött adatsort átadhatjuk a kernelnek, ahol az úgynevezett szálak kiveszik a sorszámuknak megfelelő adatokat és párhuzamosan hajtják végre ugyanazt a kernelt különböző adatokon. A szálak különböző csoportokba vannak rendezve, és egymással szinte egyáltalán nem tudnak kommunikálni, így az optimális memóriakezelés érdekében érdemes mélyebben megismerkedni a grafikus kártyák működésével.

1.3. Célkitűzések, elvégzett munka

Jelen kutatási feladat egy előre láthatóan több éves csapatmunka része, melyből a rám eső feladat egy GPU-n futtatható egyszerű neutronszimulációs kód készítése, melyen keresztül először elsajátíthatom a CUDA programnyelv sajátosságait, megismerkedhetek a GPU nyújtotta lehetőségekkel és korlátokkal, majd kipróbálhatok néhány reményteljes ötletet az optimalizáció tekintetében. Hamar kiderült, hogy a megfelelő futáshoz (stabilitás, statisztikus szórás és futásidő szempontjából) nem elegendő egy-egy kézenfekvő módszer használata, hanem alapjaiban más algoritmusverziókkal kell előállni.

Egy analóg és egy hasadáskor el nem ágazó verzióval dolgoztam a legtovább, amelyekhez igyekeztem olyan módszereket kitalálni, hogy az egyes verziók minél hosszabb valós ideig fussanak. A konvergált forrás előállítása után egy tranziens szimulációjához stabilan véges teljesítményen szükséges a rendszert tartani, ehhez növelni kell a periódusidőt, közel kritikus rendszert előállítani. A munkát nagyban megnehezíti az időfüggés vizsgálata, de éppen ez ad lehetőséget a paraméterek változásával járó tranziensek vizsgálatára.

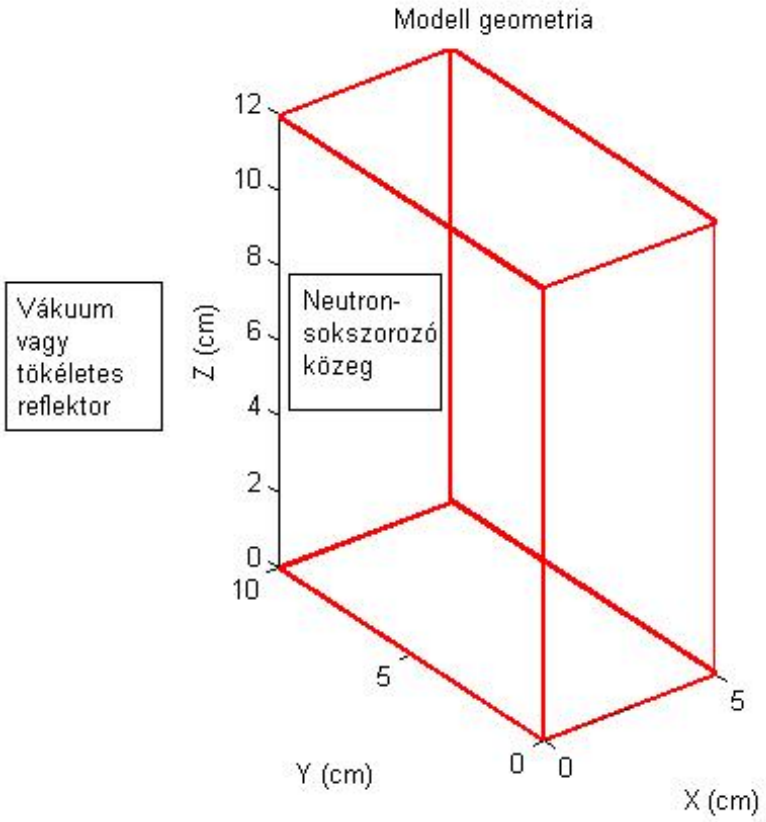
2. fejezet

Szimuláció

2.1. Geometriai és anyagi modell

A későbbiekben a feladaton dolgozó csapat egy valóságos reaktor geometriáján és paramétereivel is szeretné tranziensek szimulációjára használni a kódot, a fejlesztéshez azonban egy egyszerű téglatest geometriával dolgoztam, melyhez rendelkezésemre állt irodalom [1] a paraméterek tekintetében. A zóna körül vákuum vagy tökéletes reflektor található futástól függően, anyaga pedig homogén, izotróp neutronsokszorozó közeg. A felhasznált paraméterek a 2.1 táblázatban találhatók. A neutronok energiája alapján kétcsoport-közelítést alkalmaztam, az egyes csoportok sebessége $2{,}2 \cdot 10^5$ cm/s (0,025 eV termikus neutronok [4]) és $1{,}95 \cdot 10^9$ cm/s (2 MeV a hasadásban keletkező neutronok átlagos energiája [4]). Az egy hasadásban felszabaduló energia értéke 200 MeV [4], és hasadást kizárólag a termikus csoportban lévő neutronok okozhatnak, de a két energiacsoport paraméterei ezen kívül megegyeznek, tehát a hatáskeresztmetszetek azonosak. Későneutronok tekintetében egy csoportot vettem figyelembe, ahol közös átlagolt bomlási állandót és összegzett későneutronhányadot alkalmaztam.

A hatáskeresztmetszet 1 1/cm-re lett normálva, ehhez próbáltam olyan további hatáskeresztmetszeteket beállítani, melyek használatával leginkább megközelíthetem a kritikus rendszert. Így az átlagos szabad úthossz szintén 1 cm. Cél, hogy a szabályozórudak mozgásának mintájára az abszorpciós hatáskeresztmetszetet először pillanatszerűen, majd valamilyen folytonos függvény szerint változtatva a tranziens viselkedését lehessen elemezni, melyhez szükség lesz a kritikusságon kívül jól kezelhető reaktivitás értékek beállítására is.



2.1. ábra. Modell geometria

2.2. Szimuláció menete

Az egyes algoritmusok jelentősen eltérnek egymástól szálszervezésben, azonban az alapvető fizikai kölcsönhatásokat azonos módon kezeltem minden esetben. A Monte Carlo kódok többségéhez hasonlóan ebben a szimulációban is az egyes részecskék (neutronok) életútját kísértem végig, ahol a hatáskeresztmetszeteknek megfelelő valószínűséggel sorsoltam az eseményeket. Célom elsősorban a **teljesítmény időfüggésének** vizsgálata volt, különböző időlépések alkalmazásával. Emellett természetesen vizsgáltam ezen adatok statisztikus szórását is. A tranziens jelenségeket érdemes kis időlépések mellett vizsgálni, de a prompt neutronok keletkezésének karakterisztikus ideje 10^{-6} s, így ez alsó korlátot szab az időlépések hosszának. A mintát, vagyis az indított neutonok számát igyekeztem nagynak választani, az egyes algoritmusokhoz tartozó futásidőbeli korlátoknak megfelelően. Ilyen egyszerű geometriára és maximum néhány másodperc valós időre a programok legfeljebb néhány perc alatt lefutnak.

2.1. táblázat. A felhasznált anyagi és geometriai állandók [1][2]

Mennyiség	Jelölés	Érték
Hosszúság	x	5 cm
Magasság	y	10 cm
Szélesség	z	12 cm
Teljes hatáskeresztmetszet	Σ_t	1 1/cm
Abszorpciós hatáskeresztmetszet (közel kritikus állapot)	Σ_a	0,165 1/cm
Hasadási hatáskeresztmetszet	Σ_f	0,25 1/cm
1 energiacsoport sebessége (termikus)	v_1	$2{,}2 \cdot 10^5$ cm/s
2 energiacsoport sebessége (gyors)	v_2	$1{,}95 \cdot 10^9$ cm/s
Későneutron-hányad	β	0,00685
Átlagos bomlási állandó	λ	0,0784 1/s
Egy hasadásban felszabaduló energia	Q	$3{,}18 \cdot 10^{-11}$ J

2.2.1. Kezdeti eloszlás

Első lépésben kezdeti eloszlást igyekeztem meghatározni, amelyhez a téglatesten belülre **koszinuszos** eloszlás alapján sorsoltam a neutronok induló helyét (2.2 ábra). A választás alapja a Helmholtz-egyenlet (2.1) téglatest geometiára vonatkozó megoldása.

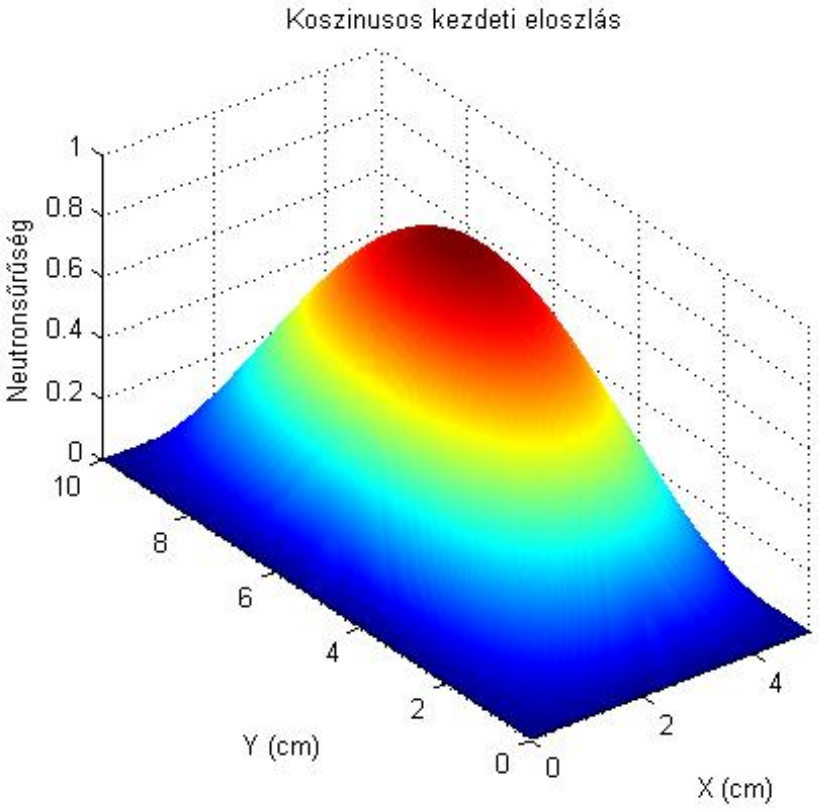
$$\Delta\Phi(\boldsymbol{r}) + B^2\Phi(\boldsymbol{r}) = 0, \tag{2.1}$$

$$\Phi_{nlk}(x,y,z) = sin(\frac{n\pi}{a}x)sin(\frac{l\pi}{b}y)sin(\frac{k\pi}{c}z) \qquad (n,l,k = 1{,}2{,}\ldots), \tag{2.2}$$

ahol a,b,c a téglapest oldalhosszai. A reaktor belsejében végig pozitív alapmódus a $\Phi_{111}(x,y,z)$ sajátfüggvény [4]. Természetesen ez az eloszlás közelít az egyensúlyi eloszláshoz, azonban nem tökéletes így a szimuláció egy tranzienssel fog kezdődni, amely konvergál az egyensúlyihoz. A későbbiekben, amikor megfelelő mennyiségű adat áll rendelkezésre, majd egy ilyen megfelelően konvergált eloszlással fogom indítani a szimulációt.

A neutronok fele-fele arányban kerülnek a két energiacsoport egyikébe, az erőforrásokkal való takarékoskodás érdekében nem sorsolva, hanem a helyet meghatározó szál indexének paritása alapján.

Azokban az esetekben, amikor vákuum helyett tökéletes reflektor vette körbe a rendszert, a koszinusos eloszlás helyett **sík** kezdeti eloszlást alkalmaztam.



2.2. ábra. Koszinuszos kezdeti eloszlás (x és y irányban)

2.2.2. Szabad úthossz, izotróp irány, reakció sorsolás

A kezdeti eloszlás meghatározása után az egyes részecskék **szabad úthossz**át inverz eloszlásfüggvény módszerével sorsoltam, amely a homogén közeg miatt:

$$s = -\frac{1}{\Sigma_t} ln(1 - r), \tag{2.3}$$

ahol r kanonikus véletlenszám.

Izotróp módon sorsoltam **irányt** minden részecskének, egy z koordináta, valamint egy szög sorsolásával a következő módon:

$$w = 2r_1 - 1 \tag{2.4}$$

$$\varphi = 2\pi r_2 \tag{2.5}$$

$$h = \sqrt{1 - w^2} \tag{2.6}$$

$$v = h \sin \varphi \tag{2.7}$$

$$u = h \cos \varphi, \tag{2.8}$$

ahol r_1, r_2 kanonikus véletlenszámok, u, v, w az irányvektor komponensei. Ehhez szögfüggvényeket kell számítani, ami köztudottan hosszadalmas folyamat, azonban GPU-n átlagosan gazdaságosabb lehet, mert egy rejekciós módszer könnyen feltarthat egyszerre több szálat is.

Az irány és a szabad úthossz ismeretében megváltoztattam a részecske helykoordinátáit.

A reakciók sorsolásához először meg kell vizsgálni, hogy a részecske benne van-e még a sokszorozó közegben? Amennyiben nincs (kiszökés), úgy nem kell vele tovább számolni. Tökéletes reflektor használata esetén kiszökés nem fordul elő, a részecske visszapattan a zóna határáról. Ha a részecske a zónában marad, akkor a szabad úthossznak megfelelő pontban a részecskével valamilyen reakció történik. A reakció lehet **abszorpció**, **hasadás** vagy **izotróp szórás**.

Abszorpció esetén vagy kihal a részecske (analóg), vagy implicit befogást alkalmaztam, hogy csökkentsem a nulla súlyú várakozó szálak számát.

$$w = w_0\Big(1 - \frac{\Sigma_a}{\Sigma_t}\Big). \tag{2.9}$$

Izotróp szórás esetén egy új irányt kap a részecske, valamint a magasabb energiájú csoportban lévők 50 % valószínűséggel átkerülhetnek az alacsonyabb energiájú csoportba.

Hasadáskor átlagosan $\nu = 2{,}5$ neutron keletkezik, melyet a kódverziónak megfelelő módon vettem figyelembe (új részecskék bankolása, súly növekedése).

A **relatív statisztikus szórás**t minden kódverzióban a következő módon számítottam:

$$\sigma = \frac{\sum w^2}{(\sum w)^2} - \frac{1}{N}, \tag{2.10}$$

ahol N az indított részecskék száma. Ezt az értéket minden időlépésre kiszámítottam. Fontos megemlíteni, hogy mivel ez egy nem Boltzmann probléma, mert nem az egyes részecskék súlyainak minden ütközés utáni statiszikus szórását számoltam, hanem egy-egy időlépés végén összegeztem a súlyokat. A statisztikus szórás pontosabb számolásához éppen ezért szükséges volna az egyes hozzájárulások eredetét végigkísérni, így a számolt információ csupán tájékoztató jellegű.

2.2.3. Későneutronok

A kezdeti futtatások során egyértelművé vált, hogy ahogyan a valóságban, úgy a szimulációban sem szabályozható a láncreakció későneutronok figyelembevétele nélkül. Egy későneutron-csoportot szimuláltam, így a hasadásokban $\beta = 0{,}00685$ hányaddal prekurzorok keletkeznek, amelyek $\lambda = 0{,}0784$ 1/s bomlási állandóval hoznak létre későneutronokat. A későneutron életidejét logaritmikusan sorsoltam:

$$t = -\frac{1}{\lambda} ln(1 - r), \tag{2.11}$$

ahol r kanonikus véletlenszám. Ez akár a szimuláció időlépéséhez képest jelentős idő is lehet. Egy-egy ilyen hosszú életű későneutron több időlépésen keresztül is foglalhat egy-egy szálat, így ha nem alkalmazok valamilyen hatékonyságnövelő eljárást,

ez a várakozás jelentős erőforrást emészt fel. Éppen ezért a prekurzorokat érdemes lesz külön bankban tárolni, vagy a kódba építeni a kényszerített bomlás módszerét [2], amely során minden prekurzor a következő időlépésben elbomlik, továbbá súlya ennek megfelelően változik. A módszerrel kapcsolatban azonban felvetődik a kérdés, hogy mennyiben torzítja a becslést, ha éppen az abszorpciós hatáskeresztmetszet megváltoztatása környékén történik a kényszerített bomlás, hiszen a tranziensekkor a későneutronok nagy szerepet játszanak. A kérdés további vizsgálatáig ezt a hatékonyságnövelő módszert nem alkalmaztam.

3. fejezet

Algoritmusok, kódverziók

3.1. Párhuzamosítás sajátosságai

Mivel az egyes részecskék mozgása és reakciói függetlenek egymástól, így a neutronszimulációs kód párhuzamosítása első ránézésre semmilyen problémát nem vet fel. A felmerülő akadályok mind programozási és futásidő-optimalizálási kérdések. Hogyan lehet az összes számunkra szükséges információt (időlépésenként) kinyerni, úgy hogy sehol ne hagyjunk erőforrást parlagon? Hogyan lehet továbbá az időigényes CPU-GPU kommunikációt minimálisra csökkenteni? A felmerülő megoldási lehetőségek között is vizsgálni kell, hogy a becslés a valósághoz közel álljon, továbbá a kód adatkezelése megfelelő legyen, tehát több szál ne használja ugyanazon adatokat, vagy ugyanazon véletlen számsorozatot.

Két alapvető elképzelésem volt, az első az analóg verzió, amely a részecske haláláig (abszorpció, kiszökés) vagy az időlépés végéig számol, illetve a hasadásban keletkező részecskék adatai eltárolja az erőforrás felszabadulásáig, egy bankba helyezni őket. A másik elképzelés, hogy az egyes részecskékhez statisztikai súlyokat rendeltem, és a reakciónak megfelelően módosítottam ezeket. A véletlen lefutású láncok tulajdonsága, hogy különböző időben érnek véget, ha nem alkalmazunk ennek elkerülésére valamilyen módszert. Történhet kiszökés vagy abszorpció, amivel a részecske súlya olyan kicsire csökkenhet, hogy nem érdemes vele tovább számolni. Ilyenkor az adott szál nem számol csak vár, amíg a többi befejezi a futást. Más szálakon pedig felszaporodhatnak a részecskék, vagy súlyuk nagyra nőhet. Ilyenkor jó volna az utóbbi szálaktól feladatot átadni az előbbi szálaknak. A feladat a felesleges várakozás és a túl sok memóriaművelet közötti optimális megoldás megtalálása.

Kódok adatkezelése ♦ A szimulálandó részecskék adatait egy **neutron** nevű **struktúrába** rendeztem, aminek előnye, a könnyű kezelhetőségen túl, hogy ilyen módon kevésbé valószínű az adatok összekeveredése és a másolások is egy paranccsal megoldhatók. Tartozik hozzá továbbá konstruktor, így új példányok esetén nem kell az

értékek beállításával bajlódni.

A struktúra tartalmazza a részecske 3 **helykoordinátá**ját, amelyet a GPU által kedvelt és éppen ilyen célra szolgáló float3 típusban tároltam. Tartalmazza továbbá a részecskéhez tartozó **idő**t, az **enegia**csoportot, valamint a kódverziónak megfelelően a statisztikai **súlyt** is tartalmazhatja.

Az energiacsoport száma egész, a továbbiak float típusúak, amely típus használata jelentős időmegtakarítást jelent a GPU-n a double típushoz képest. Időnként szükség lehet dupla pontosságra (pl. kis számok kivonása), de a jelenlegi munkámban tökéletesen elegendő a float típus. Érdemes észben tartani, hogy a szokásosan használt függvények (szögfüggvények, logaritmusfüggvények stb.) double-t várnak, így annak érdekében, hogy a float által nyújtott időnyereséget ne veszítsük el, érdemes a típusnak megfelelő speciális parancsokkal dolgozni (pl. sinf, logf stb.).

3.2. Analóg verzió

A kód írását egy analóg verzióval kezdtem, amely mérsékelten használja ki a GPU adta párhuzamosítás lehetőségét. A forrás sorsolása, az irány, a szabad úthossz és a reakció típusának sorsolása természetesen a GPU-n történik párhuzamosan, de a hasadásokkor keletkező 2-3 neutron útja elágazik, azokat nem lehet a továbbiakban egy szálon kezelni, így a szálak visszatérnek a sorsolt értékekkel és a további feladatokat a CPU végzi. Az egyes szálakon 50% valószínűséggel keletkezik a hasadásban 2 vagy 3 új részecske, amiket el kell tárolni, amíg a szükséges erőforrás fel nem szadadul. Erre a célra egy neutronbankot használtam, amelyekbe a hasadásokban keletkező neutronokat helyeztem. Szintén a bankból lehet a kihaló részecskék helyére másikat választani. A CPU és a GPU közötti kommunikáció egyike a legidőigényesebb feladatoknak, így ez a reakciónként visszatérő megoldás jelentősen rontja a futásidőt. Amikor egy részecske meghal (kiszökik vagy abszorbeálódik), a helyére a bankból egy korábbi hasadásból származó neutron paramétereit írjuk. Ha egy részecske eléri az időlépés végét, szintén a bankba kerül, helyére pedig egy olyan, amely még nem érte azt el. Amikor minden részecske (a szálaknál és a bankban is) elérte az időlépés határát a CPU összegzi az abban az időlépésben felszabadult teljesítményt, megvizsgálja a szórását és ezeket az adatokat a standard outputon ki is írja, továbbá a későbbi grafikus ábrázoláshoz fájlba rendezi. Innen indul tehát a következő időlépés.

A kódverzió **előnye**, hogy időlépésenként futásidőben ismerjük a teljesítményt nem csupán a teljes szimuláció végén. Ez a tulajdonság lehetőséget ad különböző reaktorfizikai visszacsatolások alkalmazására is.

A kódverzió fő **hátránya**, hogy szuperkritikus rendszer esetén nagyon gyorsan növekszik a részecskék száma, melyeket egyesével kell szimulálni, így az időlépések futási idejében is jelentős növekedés figyelhető meg. További akadálya a verzió

felhasználásának, hogy nem stabil, tehát nem lehet vele hosszú (másodperces nagyságrendű) valós idejű futásokat generálni. Ennek oka, hogy precíz beállítások mellett is maximum ms-os karakterisztikus idővel a rendszer leáll vagy a teljesítmény végtelenné válik.

Reflektor ♦ A 3.3 alfejezetben tárgyalt verzió esetén a reflektor használata hozta meg a kívánt eredményt, hogy hosszú ideig közel kritikus állapotban sikerült tartani a rendszert. Ezért ezt a módszert az analóg verzió esetében is megkíséreltem használni, azonban 45 perc alatt mindössze 10^{-5} s-ot sikerült így szimulálni. Annak ellenére, hogy a futásidő és a szimulált idő nem egyenesen arányos egymással, egy 10 s hosszú szimuláció számítási ideje várhatóan több éves nagyságrendű ezzel a módszerrel.

3.3. Hasadási elágazás nélküli verzió

A hatékonyság növelésének és az eredmények összehasoníthatóságának érdekében egy újabb kódverzión kezdtem dolgozni, amelyben nem keletkeznek új részecskék, így nem szükséges bankolni. A meglévők súly statisztikai paraméterrel rendelkeznek, amelyek hasadásnál növekedhetnek. Ennél a verziónál azonban hamar belefutottam abba a problémába, hogy a legtöbb részecske súlya kiszökés vagy abszorpció által nullázódik, míg néhányuk súlya hatalmasra nő. Ez a jelenség nem tesz jót a szórásnak, továbbá ugrást okoz a teljesítményben olyan esetekben, ha éppen egy nagy súlyú részecske szökik ki, vagy idéz elő hasadást. A nullás súlyú részecskék számának csökkentése érdekében itt implicit befogást alkalmaztam, tehát abszorpció esetén a részecske súlya nem nullára változik, hanem minden részecske súlya csökken az el nem nyelődés valószínűségének mértékével. A túl nagy súlyú részecskék trajektóriáját felhasítottam, két feles súlyú részecskére.

Ennél a verziónál az időlépéseket csak a teljesítmény és a szórás vizsgálatakor vettem figyelembe, a szimuláció teljes hosszán végigment a kernel és a benne lévő szálak egy-egy részecskét annak haláláig, vagy a szimuláció idejének végéig követtek. Ezért a szimuláció ideje alatt nincs információ az aktuális teljesítményről, csak annak végeztével kerül kiírásra minden adat. Ennek hátránya, hogy nem tudhatom, hogy a hosszú futásidőnek a teljesítmény elszállása az oka vagy a nem elég hatékony működés. Annak érdekében, hogy a lehető legkevesebbet várakozzon egy szál, több részecskét kell egymás után szimulálnia. Így egy elvesztett vagy a szimulálandó idő végére ért részecske után a szálnak rögtön rendelkezésére áll egy következő. Mivel a mintának megfelelő számú szál töredéke sem indítható egyszerre, ez a módszer párhuzamosítás hatásfokából nem von le.

A kódverzió **előnye** tehát, hogy nagy teljesítmények esetén sem kell nagyobb mintával számolni, továbbá futásidőben egyelőre a legjobb.

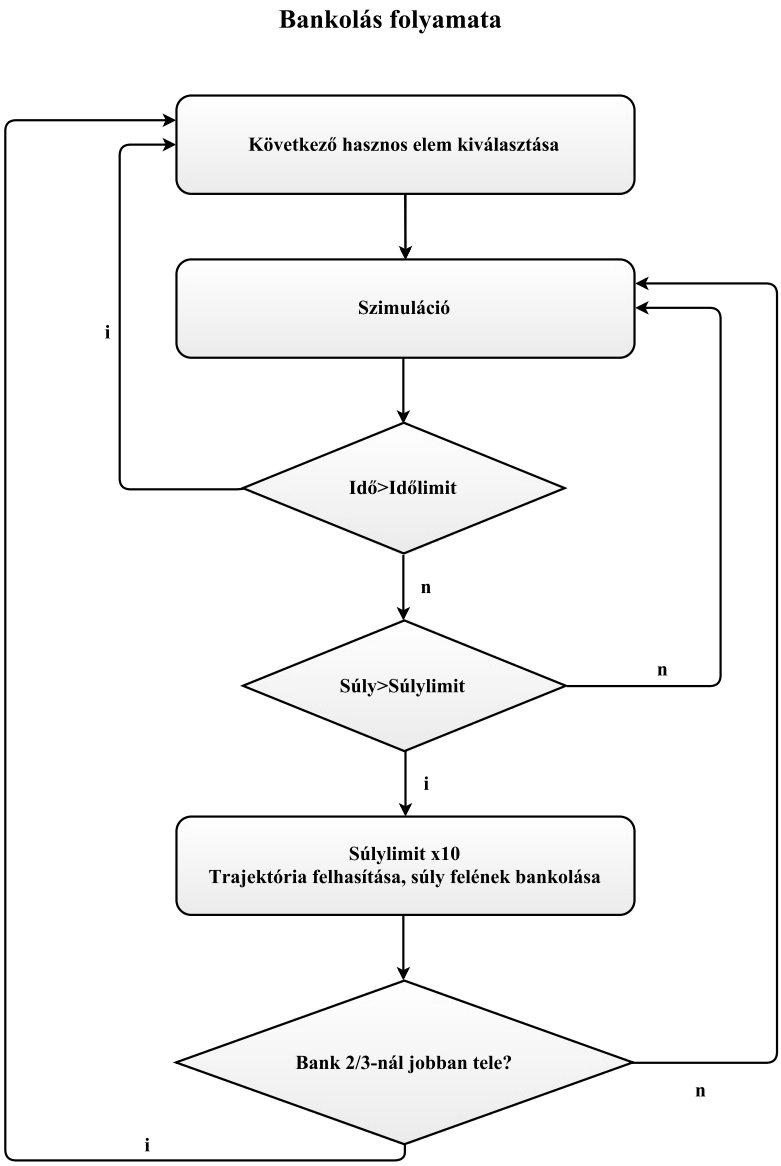
A kódverzió **hátránya**, hogy jelenlegi állapotában nem lehet reaktordinamikai visszacsatolásokat (pl. Doppler-kiszélesedés) kezelni benne, hiszen a gyorsaságát éppen az adja, hogy egy-egy részecskét a szimuláció végéig számol a következő elkezdése előtt. A későbbiekben ezt korlátozni lehet, hogy csupán egy-egy időlépés végéig számoljon minden részecskére, azonban így nagyobb bankra lesz szükség, a futásidő romlani fog. Ennek ellenére mindenképp versenyképes lesz az analóg verzióval szemben.

Trajektóriák felhasítása ♦ A felhasított trajektória egyikével lehet tovább számolni, azonban a másikat el kell tárolni egészen addig, amíg felszabadul az erőforrás a szimuláció folytatásához. Így ebben a kódverzióban is kénytelen vagyok bankot használni, annak minden nehézségével együtt.

A szálak nem kommunikálnak jól egymással és a CPU-ra bízni az újraosztást is időigényes dolog, így az algoritmust úgy szerveztem, hogy a szálak saját bankkal dolgoznak. Amint a szál végzett a rá kioszott részecskék szimulációjával, a bankban lévőkkel folytatja tovább. Ennek a banknak a méretét a szál indításakor meg kell adni. Ezen a ponton felvetődik a probléma, hogy egy túl kicsi bank hamar betelik és akkor hová tesszük a keletkező adatokat, míg egy túl nagy bank rengeteg memóriát foglal, így lassítja, vagy akár meg is akadályozza a futás indulását. Kérdés továbbá, hogy a szimuláció adott fázisában mi számít túl nagy súlynak? Hiszen a teljesítmény növelésével akár az átlagos súly is jelentősen nagyobb lehet, mint amit az az érték, amit induláskor a trajektória felhasítás határának belállítottam.

Ennek a problémának a kiküszöbölésére dinamikusan állítottam a súlyhatárt. A trajektóriák felhasításának módját és a bankolás menetét a 3.1 ábra mutatja. Hasznos elemnek a bankban olyan részecske számít, amelynek súlya nagyobb nullánál. Amint a bank 2/3-a beltelik a keletkező részecskékkel, a trajektória felhasítás határát 10-szeresére emelem, és így folytatom a szimulációt. Annak érdekében, hogy egy nagyra növő súlyú részecske miatt ne növekedjen sokáig a trajektória felhasítás határa, a bank következő elemével folytatom a szimulációt. Ha az utolsó hely is betelik, akkor nincs trajektória felhasítás annak érdekében, hogy ne vesszen el súly, a program csupán a következő elemmel számol tovább. A 3.1 ábrán bemutatott módszer alkalmazható csökkenő teljesítményre is, de ez a kód a limitcsökkentést akkor végzi, amikor végigér a bankon. Így biztosítja, hogy az aktuális limit a lehető legkisebb legyen.

Orosz rulett ♦ Ahogyan a túl nagy súlyú neutronok, úgy a túl kicsi súlyú neutronok is problémát okozhatnak a szimulációban. Kevés a hozzájárulásuk a teljesítménybe, ennek ellenére sok erőforrást kötnek le. Ezen túlmenően a szórás értékeit is rontják.



3.1. ábra. Neutronbank használatának módja a trajektória felhasításakor

Ezért egy szóráscsökkentő módszert alkalmaztam, ahol a bizonyos határ alatti súlyú részecskék valamilyen valószínűséggel meghalnak, ha nem így történik, akkor a súlyukat megnövelem [5]. A túlélés valószínűsége:

$$p = \frac{w}{w_0}, \tag{3.1}$$

ahol w_0 a módszer alkalmazásának határa. A megnövelt súly értéke nem függ a bemeneti súlytól, egy előre megadott érték, esetemben 1.

Kritikusság ♦ Minden kódverzióban problémát okoz a szimulált sokszorozó közeg kritikus közeli állapotban tartása. Nem megfelelő paraméterek esetén a reaktor

hamar leáll (nullára csökken a teljesítmény), vagy a számábrázolás szintjét is meghaladja a teljesítmény. A kritikusság közeli állapotot először a hatáskeresztmetszetek minél pontosabb beállításával igyekeztem elérni, de a statisztikus szórás meghaladja a beállítás pontosságát, így ugyanannál a beállításnál gyorsan leálló és gyorsan végtelen teljesítményű szimulációt is kaptam eredményül. Erre a problémára végül a reflektor használata adott megoldást.

3.3.1. Reflektor használata

A stabil, hosszú ideig fennmaradó kritikus állapot elérése a jelenlegi paraméterekkel és szórással vákuumos közegben a szimuláció statisztikus instabilitása miatt nem tűnt kivetelezhetőnek, ezért a modellben a vákuumot tökéletes reflektorra cseréltem. Fő problémát a nulla súlyú és a kiszökő neutronok jelentik, amelyeket ezzel a módszerrel elkerülhetünk. Ekkor a neutron visszapattan a rendszer széléről, és csak akkor hal meg, ha orosz rulettel sorsoljuk azt. Ennek eredményeképp sikerült hosszú ideig viszonylag állandó teljesítményen tartani a rendszert, a periódusidőt 10 s felé emelni. Ez a periódusidő, amely alatt egy valós reaktorban a biztonságvédelmi funkciók élesednek. A szimulációban a nagy szórás miatt ez elfogadható stabil kritikus állapotnak, így ez lehetőséget nyújt egy ebből létrejövő tranziens vizsgálatára is.

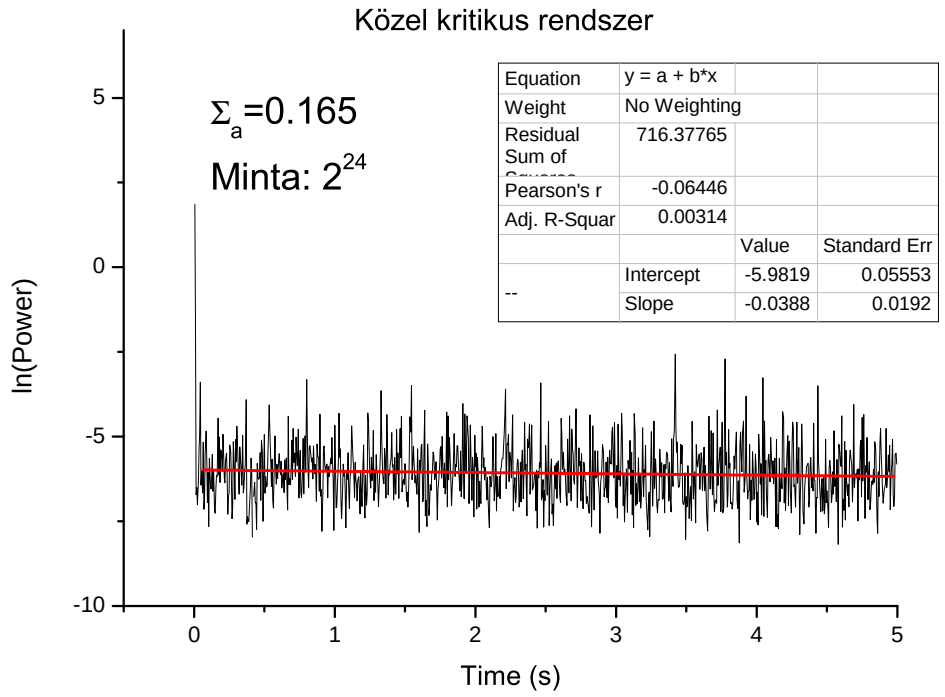
A reflektorok használatakor a koszinuszos kezdeti elolszlást sík eloszlásra cseréltem, így a forráskonvergencia időszaka lerövidül. Tökéletes reflektor esetén azt várnánk, hogy nullára rövidüljön, hiszen erre az elrendezése ez egy konvergens forrás, azonban a nulla időpillanatban egyszerre indul az összes neutron, amely jelenség torzít, tehát néhány időlépésnyit várni kell a forráskonvergenciára. A szimuláció teljes idejéhez képest ez elhanyagolható.

3.3.2. Periódusidő vizsgálata

A kritikushoz közeli állapotot az abszorpciós hatáskeresztmetszet változtatásával próbáltam elérni. Ha leállt a rendszer, megnöveltem, ha a teljesítményre végtelen értéket kaptam (tehát meghaladta a számábrázolás mértékét), akkor pedig csökkentettem. Ezzel a módszerrel sikerült 10 s-nál nagyobb periódusidőt belállítani, aminek értékét a teljesítmény logaritmikusára illesztett egyenes meredekségéből számítottam. A statisztikus szórás miatt nem érdemes a hatáskeresztmetszetet 3 tizedesjegynél finomabban beállítani.

A 3.2 ábrán látható, hogy 2^{24} (17 millió) induló neutronra átlagosan 20 %-os szórással $\Sigma_a = 0{,}165$ 1/cm abszorpciós hatáskeresztmetszet mellett 26 s-os peridódusidejű enyhén szubkritikus rendszert szimuláltam. A szimuláció stabilnak mondható, hiszen másodperceken keresztül sikerült fenntartani ezt a lapos teljesítményváltozást, azonban a periódusidő pontos értéke kismértékben futásról-futásra változik.

A forráskonvergencia időszakát, az első néhány időlépés adatait az illesztésbe nem vettem bele.



3.2. ábra. Közel kritikus rendszer beállítása

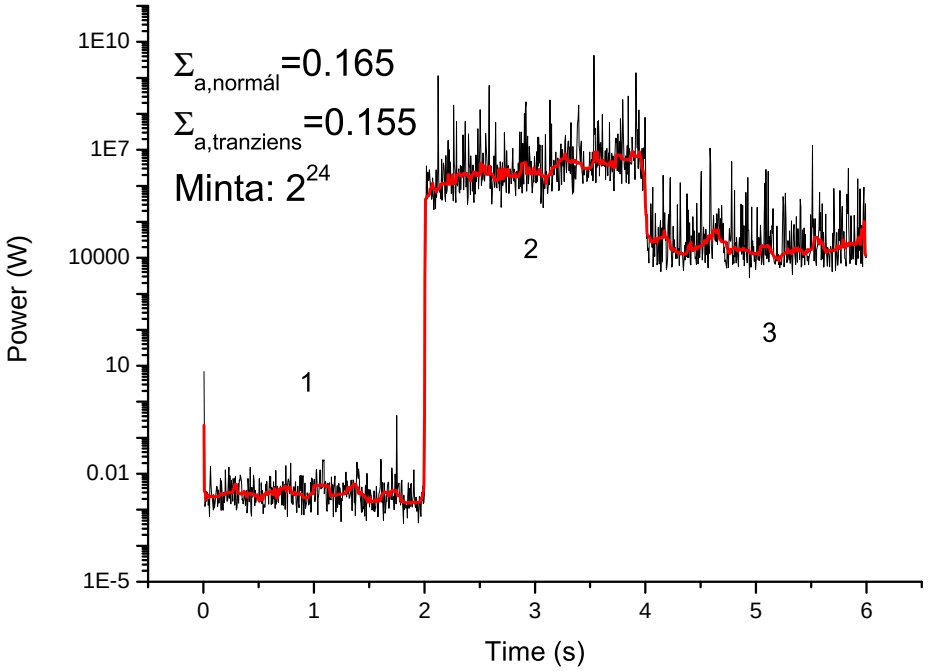
3.4. Tranziens vizsgálata

A vizsgált tranziens két reaktivitáslépcső volt, amelyet az abszorpciós hatáskeresztmetszet pillanatszerű lecsökkentésével, majd bizonyos idővel későbbi szintén pillanatszerű megnövelésével értem el. A valós reaktorban ennek analógiája a szabályozórudak fel-, majd lefele irányú mozgatása.

A várt grafikon egy konstans teljesítményről indul, majd az abszorpciós hatáskeresztmetszet csökkenésekor egy éles ugrást láthatunk, majd a $\rho > 0$ reaktivitás miatt logaritmikus skálán ábrázolva lineáris növekedést várunk, majd az az abszorpciós hatáskeresztmetszet eredeti értékre állításakor egy éles csökkenés következik, ami után a rendszer újra kritikus állapotba kerül, azonban az eredetinél nagyobb teljesítmény mellett, amelynek értéke a pontkinetikai egyenletrendszerből meghatározható.

A 3.3 és 3.4 ábrán egy kisebb és egy nagyobb reaktivitásugrás látható, melyek grafikonjaira a könnyebb értelmezhetőség kedvéért 20 pontos simítást rajzoltam.

A grafikonok jellege a vártnak megfelel, a két tranziens egymáshoz képesti viselkedését is beleértve. Tehát, ahol nagyobb a reaktivitás ugrás, ott nagyobb értékre



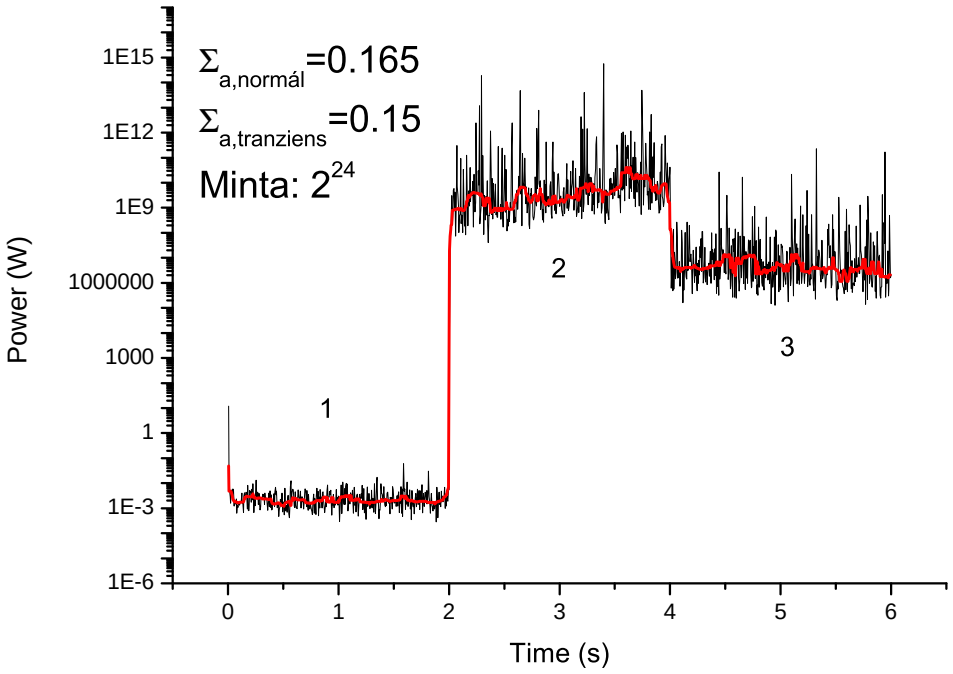
3.3. ábra. 2 s-os tranzienslépcső

3.1. táblázat. A 3.3 ábrán látható tranziens szakaszai

Szakasz	Periódusidő
1	-18,3 s
2	1,225 s
3	322,58 s

emelkedett a teljesítmény és a 2 szakasz meredeksége is sokkal nagyobb.

Érdemes továbbá megfigyelni, hogy a közel kritikus szakaszok esetében mindkét esetben (1 és 3 szakasz esetében is) láthatunk pozitív és negatív meredekséget. Ebből arra következtethetünk, hogy a szórás adta határokon belül sikerült közel az elérhető legjobb kritikusságot beállítani. Mivel a görbék jellege szemre megfelelőnek bizonyult, az ugrások nagyságának helyességét érdemes az elmélettel összevetni.



3.4. ábra. Nagyobb reaktivitásugrással járó 2 s-os tranziens lépcső

3.2. táblázat. A 3.4 ábrán látható tranziens szakaszai

Szakasz	Periódusidő
1	108,7 s
2	0,7718 s
3	-96,8 s

3.4.1. Pontkinetika

Az eredmények helyességének ellenőrzéséhez a kapott tranzienseket a pontkinetikai egyenletrendszer (3.2 és 3.3) által adott megoldással vetettem össze.

$$\frac{dP(t)}{dt} = \frac{\rho(t) - \beta}{\Lambda} P(t) + C\lambda, \tag{3.2}$$

$$\frac{dC(t)}{dt} = \frac{\beta}{\Lambda} - C\lambda, \tag{3.3}$$

ahol P a teljesítmény, ρ a reaktivitás, Λ a generációs idő, C a prekurzor koncentrációja.

A generációs időt [s] a következőképpen számítottam [4]:

$$\Lambda = \frac{1}{\Sigma_f \nu v}, \tag{3.4}$$

ahol v a neutronok sebessége. Látható tehát, hogy ez egy egycsoport modell, ahol a két energiacsoportom helyett csupán egy átlagos sebességgel számoltam.

A $\rho = 0$, kritikus állapotból induló pillanatszerű reaktivitás ugrásnak megadható az analitikus megoldása [6].

A pontkinetikai egyenletrendszer (3.2 és 3.3) mártixos felírása:

$$\frac{d}{dt}\begin{bmatrix} P(t) \\ C(t) \end{bmatrix} = \begin{bmatrix} \frac{\rho-\beta}{\Lambda} & \lambda \\ \frac{\beta}{\Lambda} & -\lambda \end{bmatrix} \begin{bmatrix} P(t) \\ C(t) \end{bmatrix}, \tag{3.5}$$

$P(0) = P_0$ és $C(0) = \frac{\beta P_0}{\Lambda\lambda}$ kezdeti értékek esetén:

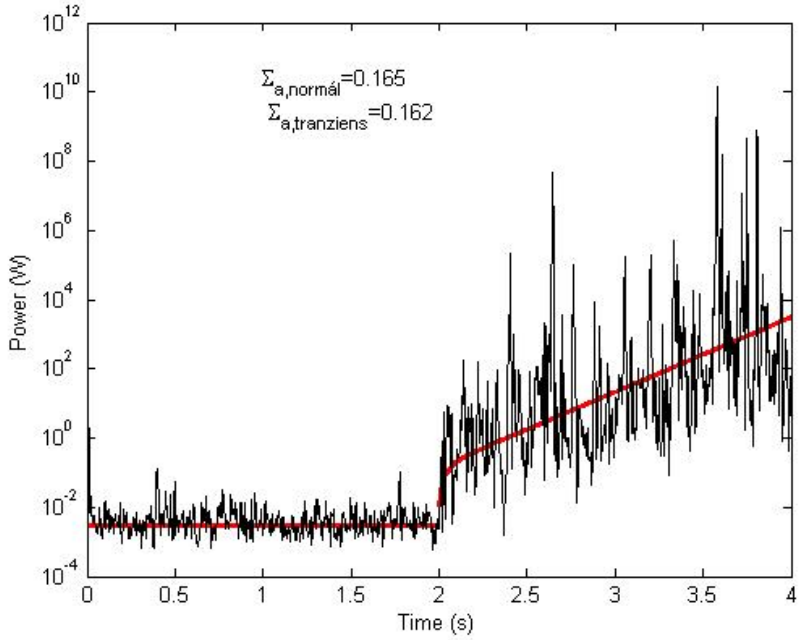
$$\begin{bmatrix} P(t) \\ C(t) \end{bmatrix} = exp\left(\begin{bmatrix} \frac{\rho-\beta}{\Lambda} & \lambda \\ \frac{\beta}{\Lambda} & -\lambda \end{bmatrix} t \right) \begin{bmatrix} P_0 \\ \frac{\beta P_0}{\Lambda\lambda} \end{bmatrix}. \tag{3.6}$$

Az egyenletrendszer megoldása a teljesítményre nézve:

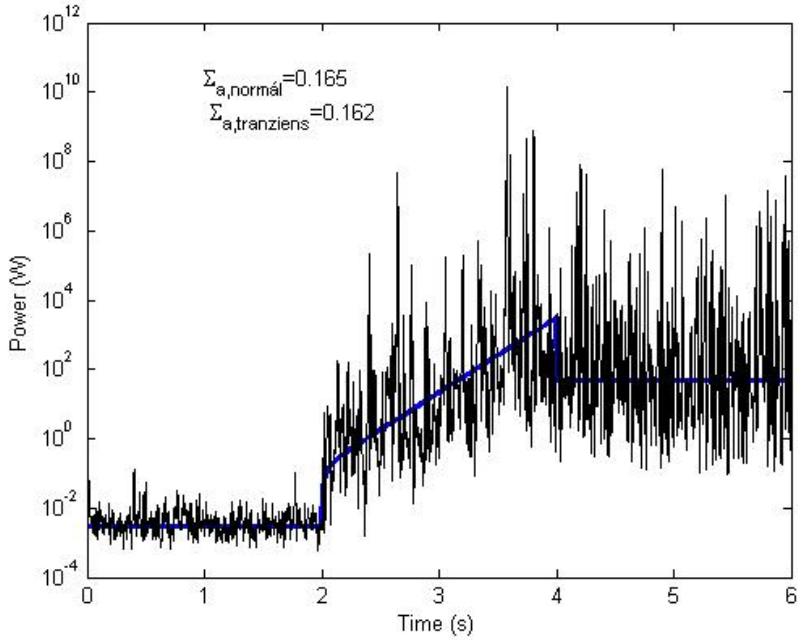
$$P(t) = P_0 \Big(\frac{\rho(exp(\omega_0 t) - exp(\omega_1 t))}{\Lambda(\omega_0 - \omega_1)} + \frac{\omega_1 exp(\omega_0 t) - \omega_0 exp(\omega_1 t)}{\omega_1 - \omega_0} \Big), \tag{3.7}$$

ahol ω_0 és ω_1 a $\begin{bmatrix} \frac{\rho-\beta}{\Lambda} & \lambda \\ \frac{\beta}{\Lambda} & -\lambda \end{bmatrix}$ mátrix sajátértékei.

A megoldás feltételezi, hogy a rendszer kritikus állapotból indul a t=0 s időpillanatban, így csupán az első két szakaszt tudom összehasonlítani az elméleti görbével. Mivel a 3.3 és a 3.4 ábrán látható mértékű teljesítményugrások nem fordulhatnak elő (ebben az esetben jelentősen károsodna a rendszer), így egy kisebb lépcsőt hasonlítottam a pontkinetikai rendszer analitikus megoldásához. A kisebb ugrás miatt, azonban ez a megoldás erőteljesebben szór. Annak érdekében, hogy a második tranzienst, az abszorpciós hatáskeresztmetszet kritikus értékre való visszaállását is megvizsgálhassam, egy numerikus megoldással is összehasonlítottam a szimuláció eredményét (MATLAB ode15s differenciálegyenlet megoldó). A legjobb illeszkedést $\rho = 0.00678$-nél, vagyis a reaktivitás 99 ¢ értékénél kaptam.



3.5. ábra. Első lépcső az analitikus megoldással



3.6. ábra. A teljes szimuláció összehasonlítása a pontkinetikai egyenletrendszer numerikus megoldásával, a nagy szórás ellenére a hasonlóság jól megfigyelhető

3.4.2. Futásidő és a statisztikus szórás kapcsolata

Felismerhető eredményeket ezzel a kóddal egy percen belüli futásidővel is lehet kapni, de a mintaszám növelésével a statisztikus szórás jelentősen csökkenthető. A 3.3 és a 3.4 grafikonokhoz 2^{24} neutront indítottam, így a 6 s szimulációjához 10-20 perc futásidő szükséges a reaktivitás változásának mértékétől függően. Ilyen módon a statisztikus szórás átlagosan 40 % alá csökkenthető.

Ehhez képest a kód kipróbálásakor 2^{15} neutront indítottam, az így kapott statisztikus szórás átlaga 90 % körül alakult.

A 3.3 és a 3.4 grafikonokon megfigyelhető, hogy a tranziens előtti statisztikus szórás sokkal kisebb, mint utána. Az első szakaszon átlagosan 20% alatti statisztikus szórás figyelhető meg, míg a második és harmadik szakasz statisztikus szórása 50% feletti értéket ad. Ennek oka a súlyspektrum torzulásában sejthető. Tehát, míg kezdetben minden részecske 1 súlyról indul, a tranziens után szélesebb lehet az eloszlás.

4. fejezet

Összefoglalás, fejlesztési tervek

Munkám során a GPU programozás és a CUDA programnyelv alapjaival ismerkedtem meg, egy neutronszimulációs kód fejlesztésének keretein belül. A Monte Carlo kód írását egészen az alapokról kezdtem a kezdeti eloszlás, a szabad úthossz és az alapvető reakciók sorsolásának menetével. Sokszorozó közeg esetén elágaznak a trajektóriák hasadáskor, így a keletkező részecskéket vagy a felhalmozódó súlyt hatékonyan kell tudni kezelni. Emellett a másik kulcskérdés stabilitás, amely jelentős kihívások elé állított. A problémát analóg módon is sikerült megoldani rövid szimulációs időre, azonban egy hasadáskor el nem ágazó, súlyparamétert használó verzióval, reflektor alkalmazásával a megfelelő paraméterek mellett a periódusidő stablian 10 s fölé emelhető 100 s-os nagyságrendű szimulációs időn keresztül.

Ez a stabilitás már elegendő egy egyszerű tranziens vizsgálatához, amilyen a reaktivitás pillanatszerű megváltoztatása. A sok nagyságrendes teljesítményugrás stabilitásához szükséges azonban a trajektória felhasítás határát dinamikusan változtatni. Eredményeimet a pontkinetikai egyenletrendszerrel való összevetéssel illusztráltam.

Fejlesztés ♦ Az eddigiekben csupán a GPU-ban rejlő lehetőségek felszínét ismertem meg, továbbá egy végletekig egyszerűsített modellel dolgoztam. A programozási módszerek, továbbá a CUDA programnyelv működésének mélyebb megértésével a számítások paraméterei optimalizálhatók, amely lehetővé teszi a valósághoz közelebb álló modell alkalmazását. A geometria pontosításán túl, a nem homogén anyageloszlású anyagokban, több energiacsoportban történő számítások is elképzelhetők. A megfelelően kidolgozott eljárással tranziensek sokasága vizsgálható különböző paraméterek mellett.

A felmerült megoldandó problémák, amelyekkel a továbbiakban dolgozni fogunk:

- Hatáskeresztmetszet folytonos változására bekövetkező tranziensek
- Hőmérsékletfüggő visszacsatolások vizsgálata
- Prekurzor bank és több későneutron-csoport figyelembe vétele

- Kényszerített bomlás módszer hatásának vizsgálata a tranziens során
- Futásidő optimalizáció
- Szóráscsökkentés
- Modell kiterjesztése több energiacsoportra
- Komplex geometria
- Heterogén anyageloszlás

Irodalomjegyzék

[1] David Legrady, J. Eduard Hoogenboom, Scouting the feasibility of Monte Carlo reactor dynamics simulations, International Conference on the Physics of Reactors "Nuclear Power: A Sustainable Resource" Casino-Kursaal Conference Center, Interlaken, Switzerland, September 14-19, 2008.

[2] Bart Laurens Sjenitzer, The Dynamic Monte Carlo Method for Transient Analysis of Nuclear Reactors, Amsterdam, 2013.

[3] CUDA Toolkit v7.5, Programming Guide.

[4] Szatmáry Zoltán, Bevezetés a reaktorfizikába, Akadémia Kiadó, ISBN 963 05 7734 8, Budapest, 2000.

[5] Ivan Lux, Laszlo Koblinger, Monte Carlo Particle Transport Methods: Neutron and Photon Calculations, CRC Press, Inc., Boston, 1991.

[6] Abdallah A. Nahla, Analytical solution to solve the point reactor kinetics equations, Tanta, Egypt, 2010.