GPU gyorsítás az AliROOT keretrendszerhez

Nagy Máté Ferenc[a,b], Barnaföldi Gergely Gábor[a] ,
Agócs András Gábor[a,b], Berényi Dániel[a,b], Bencze György[a], Boldizsár László[a], Futó Endre[a],
Hamar Gergő[a], Kovács Levente[b], Lévai Péter[a], Lipusz Csaba[a], Pochybova Sona[b] és Varga Dezső[b]

[a]MTA KFKI Részecske- és Magfizikai Kutatóintézet, 1121 Budapest, Konkoly-Thege út 29-33.
[b]Eötvös Loránd Tudományegyetem, 1118 Budapest, Pázmány P. Sétány 1/A.

Motiváció

Az elmúlt 5 év technikai fejlődése lehetővé tette, hogy egyes fizikai számításokat az eddigieknél sokszor gyorsabban el tudjunk végezni. Ez részben a játékipar rohamos fejlődésének köszönhető, ami a megjelenítő hardver számítási kapacitását a számítógép központi egységének képességei fölé helyezte. A videokártyák használatával eddig távolinak tűnő, nagy számítási kapacitású numerikus módszerek elérhető távolságba hozhatók, illetve a meglévők felgyorsíthatók. Ezt a minőségbeli javulást a grafikus kártyák képességeihez illeszkedő számításokban lehet elérni.

A kulcsfontosságú pont ami eldönti, hogy egy probléma alkalmas-e videokártya gyorsításra, az a párhuzamosíthatóság. A GPU rengeteg végrehajtóegységgel bír, amik 1-1 processzormagnak feleltethetők meg. Egy eljárás annál hatékonyabb lesz GPU gyorsítás során, mennél kevesebb az ún. shader processzorokon futó szálak között a kommunikáció. A lokális kölcsönhatások teljesen megengedettek, erre külön memóriaterület létezik az ALU egységen belül, ám a globális kölcsönhatások gyakran végzetesek az adaptációra, míg a szálak közötti gyakori és globális kommunikáció, szinkronizáció teljesen ellehetetleníti azt.

AliROOT: az ALICE szimulációs keretrendszere

Az AliROOT az ALICE kísérlet részére kifejlesztett egy fizikai szoftvercsomag, amely a ROOT háromdimenziós tervező és matematikai keretrendszerre épül. Az AliROOT tartalmaz több, a részecskefizikában használatos Monte-Carlo eseménygenerátort. Így pl. a PYTHIA6 és a HIJING generátorok adják a mikroszkópikus részecske folyamatokat. Ezek az eseménygenerátorok véletlen számok nagy léptékű mintavételezésén alapulnak.

Az eseménygenerátorok véletlen számok nagy léptékű mintavételezésén alapulnak. A véletlenszám generálás független eseményekből áll, így ezek az eljárások jól párhuzamosíthatók.

Hipotézisünk az volt, hogy sok véletlenszám időigényes generálását GPU segítségével felgyorsíthatók.

Az OpenCL API

Az OpenCL egy platform- és gyártófüggetlen szoftverfejlesztői itnerfész, amit erősen párhuzamosítható programok írására fejlesztettek ki. Általa lehetőség nyílik egyetlen programkód megírásával több, pl. Windows, Linux vagy Macintosh környezetekre fejleszteni, amiket egyaránt lehet futtatni valamennyi nagy processzor és videokártya gyártó cég termékein. Az interfész flexibilitása lehetőséget ad a programozónak arra, hogy a program optimalizálása online történjen, akár a program futása közben, figyelembe véve az éppen rendelkezésre álló hardvert. Programunkat tehát ebben a sokrétű nyelvben, OpenCL-ben írtuk.

Véletlen szám generálása GPU-n

Az összehasonlítások alapjául a ROOT csomag alapértelmezett véletlen szám generátorai szolgáltak. Ezeket az AliROOT TRandom1, 2, és 3 osztályaiban találjuk meg. A GPU gyorsította generátort egy új TRandom4 matematikai osztályként integráltuk a programcsomagba, ügyelve a felülről kompatibilitásra.

A tesztekben megvizsgáltam a generált számok eloszlását, auto-korrelációját valamint a generálási idejüket. A generálási idő a GPU-s változatnál két féle képpen értelmezhető: az egyik a teljes generálási idő, ami magába foglalja az egész keretet, ami szükséges ahhoz hogy CPU-n megtörténjen a kernel hívás, a számokat legeneráljuk majd az eredményeket visszakérjük a RAM-ba; A másik pedig a kernel idő, ami csak azt az időt foglalja magába amíg a számítás a videokártyán végbement.

Noha a GPU számítási teljesítményét ez utóbbi módon mérhetjük, vizsgálatunk fő tárgyát az első generálási eljárás képezte, amely a tényleges szimulációs időkről referál.

Véletlen számok generálása Mersenne-Twister eljárással

A Monte-Carlo eljárások során használt pszeudo-véletlen szám generátorok (PRNG) tulajdonságai kulcsfontosságúak. Nem csak a sebességük miatt kell megfontoltan választani PRNG-t, hanem azok numerikus tulajdonságai miatt is. Négy fontos tulajdonságot szokás a generátorokkal kapcsolatban megemlíteni: az eloszlás jóságát, a periódushosszat, és a memória igényét, és a fentebb említett sebesség. Nincs olyan generátor, amelyik mindegyik kritérium szerint a jobb lenne az összes többinél. A tudományos élet egyik legnépszerűbb PRNG-je a Mersenne Twister algoritmus, mely kiemelkedően jó eloszlású és hosszú periódust mutat, ugyanakkor sebességre közepesnek mondható. Ezzel áll szemben nagy memóriaigénye.

Az algoritmus a generátor magjából inicializálja a belső állapotokat, amiket újra be kell állítani annyi szám generálása után, ahány belső állapotot tárol az ember. Ez egy tökéletes példa, miként váltható a memóriaigény direkt módon sebességre. Mennél több állapotot tárol az ember, annál ritkábban kell újra inicializálni.

Jelen SFMT algoritmus előnyei közé tartozik, hogy 32 bites műveletekkel generálható 64 és 128 bites véletlen számok is, valamint kifejezetten alkalmas vektorizálásra – azaz jól párhuzamosítható. A feladathoz használt PRNG 8 belső állapotot használ, ez a videokártyákon lévő szűkös regiszter számmal magyarázható. Ez általános videokártya programozói gyakorlat, hogy erős memória optimalizációkat kell eszközölni az aránytalanul nagy shader processzor kapacitás ellensúlyozására, azaz a memóriaigényt csökkenteni, a számításigény kárára.

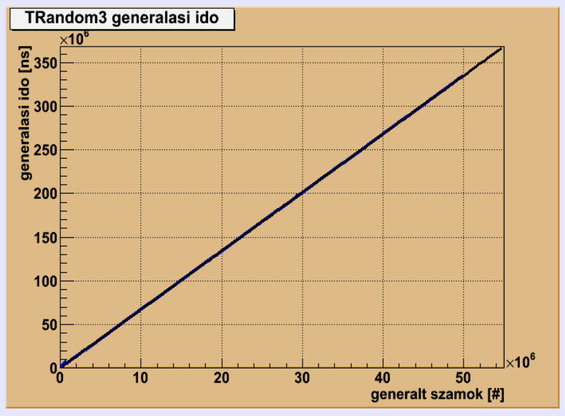
Eredmények: a véletlenszám-generátor jósága

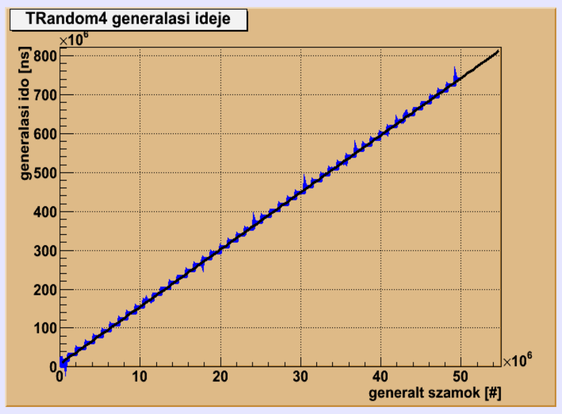
A kapott generálási statisztikák alapján az általunk írt GPU-s TRandom4 generátor kellően jó a CPU-n futó TRandom referencia generátorokhoz képest. A generált véletlen számokat az 1. grafikon láthatjuk. Az átlagok, amelyeket 1. táblázat mutat, mindegyik esetében hasonlóan közel állnak a várt 0,5 értékhez. Eredményünket a generált számok auto-korrelációja is alátámasztja: ez a csak a 0 értéknél kimagaslóan nagy (közel 3 nagyságrendnyire emelkedik ki valamennyi korrelációs távolság közül ld. 2. grafikonok).

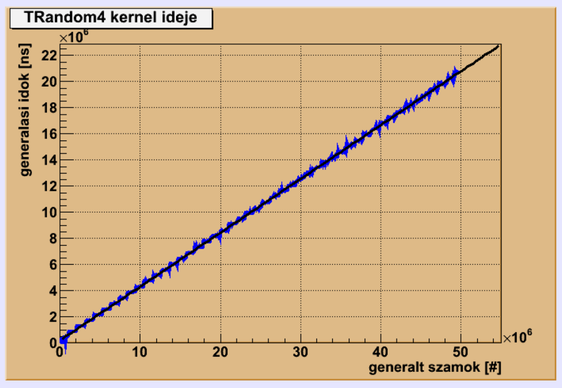
A generálási időt vizsgálatához a szintén Mersenne-Twister algoritmust használó TRandom3-at hasonlítottuk össze a GPU-s TRandom4 osztállyal. A 2. táblázatban összegeztük eredményeinket: a GPU-s TRandom4-nél értelmezhető kétféle generálási idő nagysága, közrefogja a referencia értéket. A teljes generálási sebesség közel kétszer lassabb, mint a tényleges generálási idő, ami ellenben több mint 16 gyorsabb a CPU-s változathoz képest (3. grafikon).

Mért idő	Egyenes meredekség	Egyenes tengelymetszet	Generálási idő [ns / db]	Ráta [db / ns]
TRandom3 kernel	6,697	$3,7*10^5$	6,697	0,14932
TRandom4 kernel	0,411	$2,0*10^5$	0,411	2,43309
TRandom4 teljes	14,7	$7,7*10^6$	14,7	0,06802

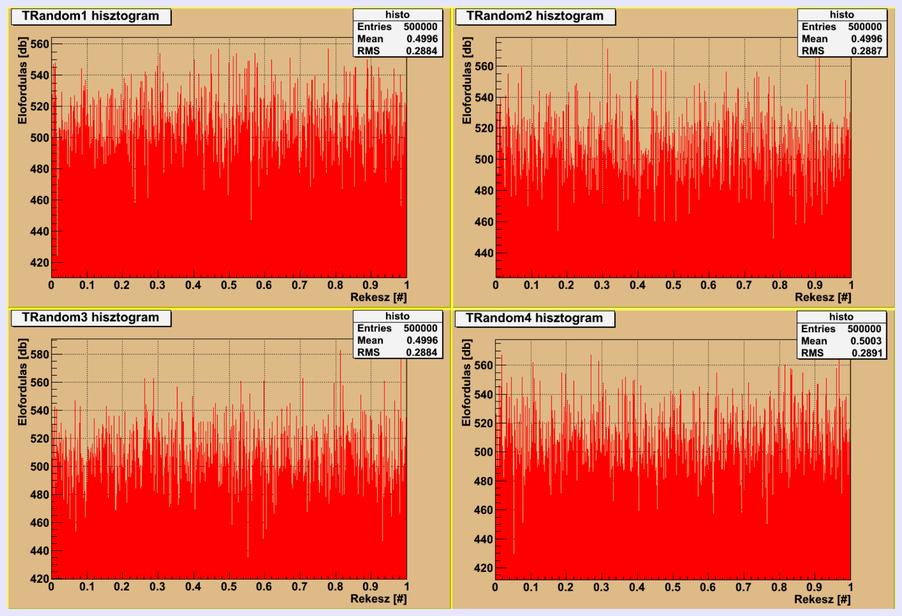
2. táblázat



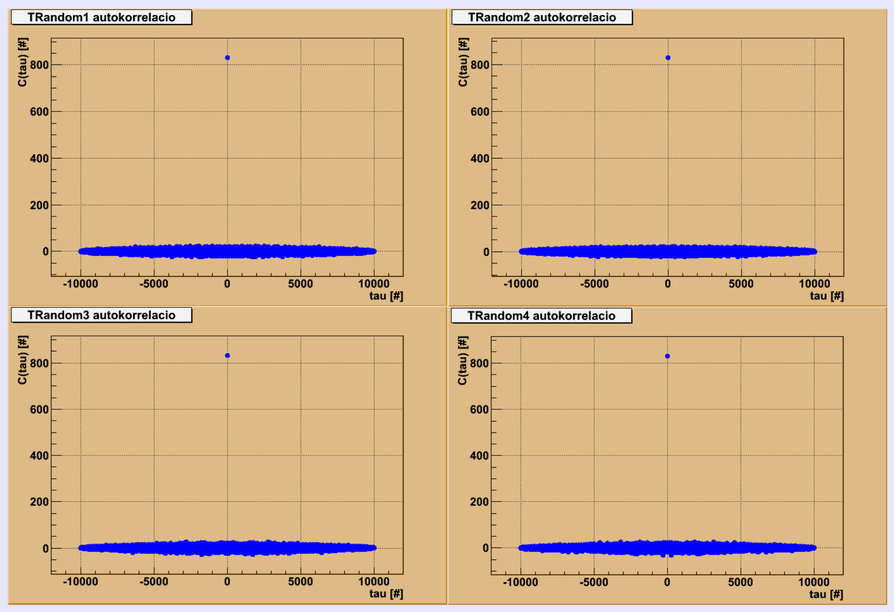




3. grafikon



1.grafikon



2. grafikon

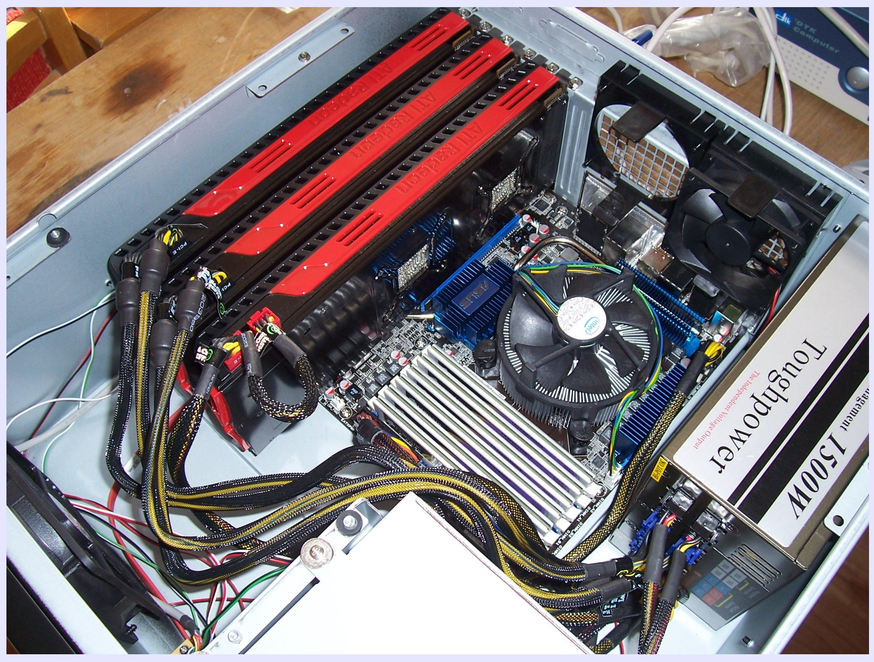
1. táblázat

Generátor	Átlag	Gyök négyzetátlag
TRandom1	0,4996	0,2884
TRandom2	0,4996	0,2887
TRandom3	0,4996	0,2884
TRandom4	0,5003	0,2891

Tesztkörnyezet

A tesztek szempontjából fontos volt, hogy azonos kategóriájú alkatrészeket hasonlítsunk össze teljesítményben, ezért a tesztgép egy Intel Core-i7 2,66GHz-es processzorból, 12GB 1333MHz-es DDR3 RAMból állt, amik 3 db ATi Radeon 5970-et hajtottak meg.

A fejlesztői környezet korlátozásai miatt a véletlen generátor 1 GPU magot tudott hasznosítani a rendelkezésre álló 6-ból, ám mivel a referencia CPU változatok is egy szálon dolgoznak csak, ezért ez nem okozott gondot.



*VHMPID kollaboráció: Bari-Budapest-CERN-Chicago-Mexico-New Haven-Pusan

A témát az OTKA NK77816, NKTH-OTKA CK 77719, NKTH-OTKA CK 77815, PD 73596 számú pályázatok, valamint a Bolyai János Kutatási Ösztöndíj támogatták.

A Large Ion Collider Experiment
European Organisation for Nuclear Research