HADMÉRNÖK

IV. Évfolyam 4. szám - 2009. december

Szénási Sándor
szenasi.sandor@nik.bmf.hu

GPGPU – ÚJ ESZKÖZÖK AZ INFORMÁCIÓBIZTONSÁG TERÜLETÉN

Absztrakt

A processzor architektúrák elmúlt években bekövetkező fejlődésének legfőbb iránya a feldolgozóegységek számának növelése, ez a többmagos processzorok megjelenésén túlmenően egészen újszerű architektúrák kialakulását is jelenti, mint például az általános célú számításokra használható grafikus kártyák. Ezek az eszközök kiemelkedő ár/teljesítmény aránnyal rendelkeznek, az általuk kínált elméleti csúcsteljesítmény azonban csak megfelelően tervezett és implementált algoritmusok használata esetén aknázható ki hatékonyan. A gyakorlatban felmerülő problémák esetében ez a teljesítmény így gyakran nem, vagy csak részben használható ki, ennek vizsgálatát mutatja be ez a tanulmány.

In the last few years, the general trend in processor development has been to increase the number of processing units. This means not only to introduce multi-core processors, but also to develop patent architectures like GPGPUs (General Purpose computation on Graphics Processing Units). The cost/benefit ratio of these devices is outstanding, but their peak performance can be achieved by only properly designed and implemented algorithms. So in practice this performance usually not (or not fully) achievable, this is the main topic of this article.

Kulcsszavak: *GPGPU, CUDA, párhuzamos algoritmusok, teljesítmény mérések ~ GPGPU, CUDA, parallel algorithms, performance benchmarks*

PÁRHUZAMOSÍTÁS KIKÉNYSZERÍTÉSE

Az informatikában megszokottá vált, hogy a különböző hardver eszközök, köztük a processzorok teljesítménye évről-évre folyamatosan növekszik, miként ezt a közkedvelt Moore-törvény is kimondja. Ez a nagyon egyszerű jóslat hosszú éveken keresztül jónak bizonyult, napjainkban azonban egy érdekes változásnak lehetünk tanúi. A processzorgyártók néhány évvel ezelőtt kénytelenek voltak szembesülni azzal, hogy nem tudják az órajel-frekvenciákat az eredetileg elképzelt ütem szerint növelni, ami miatt új utakat kellett keresni a további fejlesztések számára. Ezek közül az utóbbi években a legszembetűnőbb a többmagos processzorok megjelenése, amelyek az évenkénti teljesítménynövekedést egy meglehetősen

kézenfekvő ötlettel biztosítják: egy processzoron belül bizonyos egységeket egyszerűen megdupláznak, amelynek segítségével az eszköz elméleti számítási kapacitása is duplázódik.

Ezzel a piac által elvárt szükséges növekedés tehát igazolható, és természetesen nincs szó csalásról, ez a teljesítménynövekedés valóban elérhető és kézzelfogható. Azonban a gyakorlati felhasználás tekintetében már fontos megszorításokkal kell szembenéznünk, ha hozzá akarunk férni a megnövekedett számítási kapacitáshoz. Hagyományosan, ha csak a legnagyobb értékesítési területet, a személyi számítógépek piacát tekintjük (ahol a fejlesztők számára elengedhetetlen, hogy minden újonnan megjelenő eszköz kompatibilis legyen elődeivel), a processzorok teljesítményének növelése azonnali, közvetlen gyorsulást jelentett az alkalmazások futtatása során. Egy kétszer nagyobb teljesítményű processzor telepítése, az alkalmazások bármiféle módosítása nélkül is körülbelül kétszer rövidebb futási időt eredményezett. A jelenlegi trend esetén azonban ez már nem ennyire magától értetődő, egy manapság elérhető nagyteljesítményű kettő vagy négymagos processzoron futtatva a hagyományos alkalmazásokat valószínűleg nem érhető már el azonnali kétszeres vagy négyszeres gyorsulás. Sőt, annak is meglehetősen nagy az esélye, hogy a processzor magok számának növelésével a program futási ideje szinte egyáltalán nem változik (leszámítva némi közvetett gyorsulást, amely annak köszönhető, hogy az egyszerre több feladatot végrehajtó operációs rendszer ilyenkor egy teljes processzormagot szentelhet az említett feladatnak).

A processzorgyártók tehát ezzel a lépéssel a szoftverfejlesztőkre hárítottak meglehetősen sok megoldandó problémát, amennyiben a nagyobb számítási kapacitást ki akarják használni. Egyrészt a programozók hagyományosan nem párhuzamos algoritmusokat fejlesztettek/fejlesztenek. Mivel éveken keresztül általánosnak tekinthető volt az egy darab vezérlő egység használata (természetesen régen is léteztek többprocesszoros rendszerek, de ezek szerepe jóval kevésbé volt hangsúlyos, mint napjainkban), ezért a programozók is erre optimalizálták algoritmusaikat. Ezzel az alkalmazásfejlesztés egyszerűbb, gyorsabb, áttekinthetőbb lett. Napjainkra azonban látható, hogy a processzorok gyorsítása már főleg a párhuzamosítás irányában képzelhető csak el, így ennek megfelelően a programokat is adaptálni kell ehhez az új környezethez, ehhez egyelőre még hiányzik a szakértelem és tapasztalat.

Ennél nagyobb problémát jelent az, hogy vannak feladatok, amelyek jellegükből adódóan nem párhuzamosíthatók. Egy hosszabb számítási sorozat, amelynek minden művelete függ az előző műveletek (rész)eredményeitől, csak egymást követő lépések sorozataként számítható ki, ami mindösszesen egy végrehajtóegységet igényel. Hiába van több, az ezek adta lehetőség nem használható ki. Emiatt az egyszerű programozási technikák megújításán túl szükség van egy sokkal nagyobb paradigmaváltásra is, az eddig ismert és gyakorlatban jól alkalmazott szekvenciális algoritmusok módosítására, illetve új, párhuzamosíthatóbb eljárások keresésére.

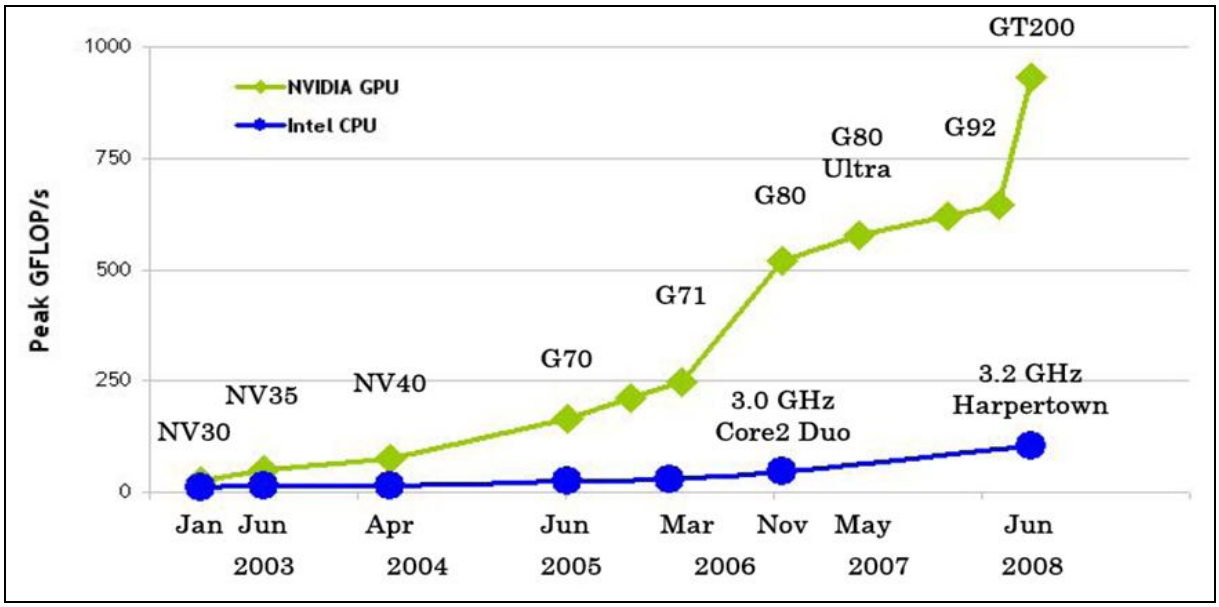
ÁLTALÁNOS CÉLÚ GRAFIKAI FELDOLGOZÓ EGYSÉGEK

Számítási kapacitás tekintetében általában a számítógép központi végrehajtóegységének teljesítményét vizsgáljuk, az utóbbi években azonban egy újszerű lehetőséggel is számolnunk kell, ezek pedig a személyi számítógépekben található grafikus kártyák.

Ezek az eszközök hagyományosan a monitoron látható kép megjelenítéséért felelősek. A múltban ez meglehetősen egyszerű feladatként jelent meg: a memória egy bizonyos területén lévő tartalomnak átalakítása analóg/digitális kimenetté, a 90-es években azonban itt is lezajlott egy nagyszabású változás. A számítógépes játékipar (és részben a tervezői

rendszerek) igényeit kielégítve megjelentek az úgynevezett grafikai gyorsító kártyák. A videojátékok megvalósításukat tekintve mind egymástól független alkalmazások, azonban a háromdimenziós megjelenítés tekintetében nagyjából hasonló algoritmusokat alkalmaznak. Mivel egy játékprogram erőforrásigényét tekintve ez igen tekintélyes részt jelent, megjelentek úgynevezett kiegészítő kártyák, amelyek célirányosan ezt a 3D számítást segítették, ezzel jelentősen csökkentve a központi egység terhelését.

Ezek a kiegészítő kártyák először fizikailag külön kártyaként jelentek meg, a videokártyához kapcsolódóan, de hamarosan a két eszközt integrálták, ennek következtében a grafikus kártyák az egyszerű jelátalakításon túl hamarosan meglehetősen nagy számítási kapacitással rendelkező kiegészítő eszközökké váltak. Kezdetben ezek céljuknak megfelelően különböző, a 3D számításokhoz szükséges számításokat elvégző részegységeket tartalmaztak, a későbbiekben azonban folyamatosan növekedett ezen egységek száma, illetve az utóbbi években ezen egységek felépítése is egyre hasonlóbbá vált, mígnem a közelmúltban megjelentek az első, azonos felépítésű feldolgozóegységeket tartalmazó eszközök.



1. ábra: CPU – GPU csúcsteljesítmények változása [1]

Ez a lépés pedig egy egészen újszerű helyzetet teremtett az alkalmazás fejlesztők számára. Az így létrejövő eszközök ugyanis már általános célú feldolgozó egységeket tartalmaznak, így bár a grafikus kártyák fő feladata továbbra is a kép megjelenítése maradt, megjelent a lehetőség, hogy ezeket az eszközöket egyéb, általános célú programok futtatására használjuk. Hamarosan a grafikus vezérlőkártya gyártók is meglátták az ebben rejlő lehetőségeket, és eszközeiket átalakították olyan formában, hogy egyszerű módon támogassák az általános célú alkalmazások futtatását. Az új technológia hangzatos nevet is kapott: GPGPU – General-Purpose computation on Graphics Processing Units.

Ki kell hangsúlyoznunk, hogy az új lehetőség nem csak technikai érdekességként van jelen, a napjainkban kapható videokártyák ugyanis már akár több száz végrehajtó egységet is tartalmazhatnak, ezeket használva az alkalmazások futásideje akár két nagyságrenddel is kisebb lehet. Az 1. ábrán látható az évek tükrében a CPU-k (központi feldolgozó egység) és GPU-k (grafikus vezérlő) sebességének növekedése. Jól látható, hogy az évek folyamán a GPU-k nagyságrendekkel nagyobb teljesítményre tettek szert, mint korunk legmodernebb CPU-i.

ELÉRHETŐ ELMÉLETI TELJESÍTMÉNY

Azonban a gyakorlati megvalósítás tekintetében már fontos megszorításokkal kell szembenéznünk: ha hozzá akarunk férni a megnövekedett számítási kapacitáshoz, ez csak a nagyszámú végrehajtó egység kihasználásával, tehát megfelelően párhuzamosítható algoritmusok tervezésével, és ezek megfelelő implementálásával érhető el, amelyek a már előzőleg is említett akadályok miatt egyelőre nehézkesnek tűnnek.

Mindez a jelenlegi videokártya piacra pedig hatványozottan érvényes. A grafikus vezérlőkártyákban a hagyományos CPU-khoz képest meglehetősen egyszerű végrehajtó egységekkel találkozhatunk (utasításkészlet, sebesség, optimalizálási lehetőségek tekintetében). Ezek ereje tehát csak nagy számuknak köszönhető, ez azonban igényli azt, hogy egy feladat esetén ezek az egységek mind kihasználhatóak legyenek, tehát nagyon sok szálon párhuzamosan futtatható programokra van szükség. Érdemes tehát néhány gyakorlati vizsgálatot elvégezni, hogy ezek az eszközök a való világ feladataiban mennyire tudják megközelíteni az elméletben ígért teljesítményüket.

Az egyes GPGPU-k elméleti teljesítménye meglehetősen egyszerűen kiszámolható. Tekinthetjük példaként az nVidia cég egyik nagyteljesítményű eszközét, a GTX280-as videokártyát. Ennek alapvető adatait tartalmazza az 1. táblázat:

Mag típusa:	GT200
Megjelenés ideje:	2008 jún.
Feldolgozó egységek száma	240
Shader órajel-frekvencia	1,296 Ghz
FP32 utasítás/órajel	3
Memória órajel-frekvencia	2214 Mhz
Memória interfész	512 bit

1. táblázat [2]

Ez alapján egyszerűen kiszámítható az elméleti számítási kapacitás. A magok órajel-frekvenciája 1,296 Ghz, illetve ideális esetben egy órajel-ciklus alatt 3 darab lebegőpontos művelet végrehajtására képesek. Mivel az említett videokártya 240 darab egymástól független végrehajtó egységet tartalmaz, így a tényleges maximális teljesítmény ezek szorzata, tehát 933,12 GFLOPS. Ez messze meghaladja a jelenleg kereskedelmi forgalomban kapható átlagos eszközök által elérhető teljesítményt.

Mivel az általános programozási feladatok az egyszerű számítások mellett nagymennyiségű memória műveletet is szoktak tartalmazni, célszerű megvizsgálni a kártya ezen tulajdonságait is. A szóban forgó GTX280-as kártya esetén a memóriavezérlők 2214 Mhz órajel-frekvencián működnek, sávszélességük pedig 64bit. 8 darab ilyen memóriavezérlő található az eszközön, így ebben az esetben is egyszerűen számítható az elérhető maximális átvitel: 1133,57Gb/s. Tehát ideális körülmények között másodpercenként 141,7 GB adat átvitelére alkalmas.

Fontos azonban megjegyezni, hogy mindezek a rendkívül impozáns adatok pusztán elméleti határokat jelenteket (a gyártók által megadott adatokra alapozva), ezt a tényleges teljesítményt a gyakorlati alkalmazások valószínűleg soha nem fogják elérni, csak külön erre a célra írt tesztprogramokkal lehet megközelíteni (ahol az utasítások jellege és sorrendje éppen a kártya számára legoptimálisabb).

Emiatt fontos különböző gyakorlati teszteket végezni valós problémákkal, amelyek segítenek meghatározni a kártya való világban kihasználható teljesítményét, amelyek alapján sokkal egyszerűbb döntéseket hozni a gyakorlati használhatóság területén.

MÁTRIX-SZORZÁS VIZSGÁLATA

Az előzőekből kiderült, hogy az új eszközök használata nem ad automatikus sebességnövekedést, egy hagyományos program futtatása körülbelül hasonló eredményt adna, mint az előző generációs processzorok esetén. Valódi előny kimutatásához célszerű valamilyen jól párhuzamosítható algoritmust vizsgálni, ilyeneket a matematikai feladatok között meglehetősen gyakran találunk, tekintsük példaként az egyszerű mátrix szorzás algoritmusát, ahol az egyes részeredmények kiszámítása egymástól teljesen független.

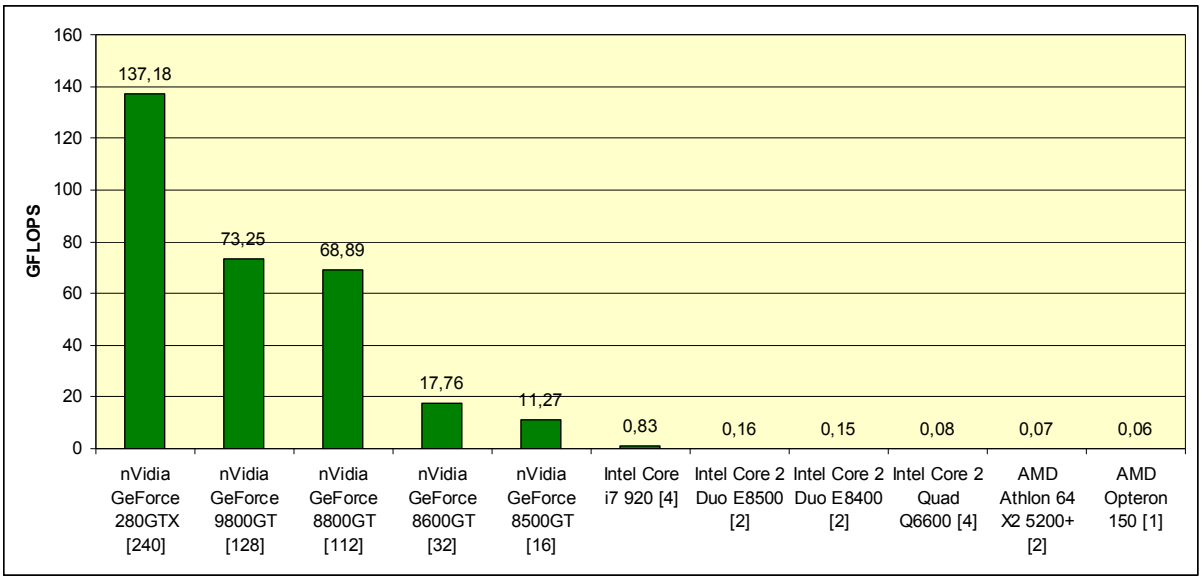
Egyprocesszoros rendszer esetén a megoldás két egymásba ágyazott ciklust igényelne, amelyek segítségével kiszámítható egymás után minden sor és ezen belül minden oszlop értéke (minden elem számítás egy harmadik ciklust igényel, amely összeszorozza, majd összeadja a bemeneti mátrixok elemeit).

A párhuzamosítás itt meglehetősen egyszerűen megoldható, platformtól függően különféle megoldásokat célszerű alkalmazni. Többmagos CPU-k esetében a magok száma nagy valószínűséggel alacsonyabb, mint a feldolgozandó mátrix elemeinek a száma, emiatt valamilyen formában ki kell osztani az egyes egységek között a feldolgozandó mátrix elemeket. Az egyik lehetőség, hogy a feldolgozó szoftver a magok számával megegyező számú szálat indít el, majd ezek számára kijelöli a mátrix megfelelő szegmenseit. Az egyes szálak által végrehajtott algoritmus itt tulajdonképpen megegyezik az egyprocesszoros rendszernél megismerttel (három egymásba ágyazott ciklus), csak minden szál a teljes mátrixnak csak egy részén dolgozik. Másik lehetőség, hogy a mátrix elemeinek számával megegyező számú szálat indít a program, minden szál csak egy elem kiszámításával foglalkozik. Ebben az esetben az egymással párhuzamosan futó szálak fizikai végrehajtóegységekre való felosztását az operációs rendszer fogja végezni valamilyen ütemezési stratégia szerint. Mivel a szálak ütemezése meglehetősen időigényes folyamat, valószínűleg az előző megoldás vezet jobb eredményre.

A GPGPU-k programozása esetén azonban más megközelítéssel célszerű megoldani a feladatot. A feldolgozó egységek meglehetősen nagy száma, illetve a keretrendszer sajátosságai miatt (adatpárhuzamos végrehajtás esetén tudnak csak hatékonyan együttműködni a feldolgozóegységek) a második változat használható jól.

A gyakorlati megvalósítás során használt CUDA keretrendszer esetén nincs is lehetőség az egyes szálak közvetlen felparaméterezésére, a program indítása tulajdonképpen kimerül a feldolgozandó, illetve az eredmény mátrixot tartalmazó memóriaterületek azonosításában, az elindítandó szálak programkódjának (kernel), illetve indítási paramétereinek meghatározásában.

A szálak létrehozását, azok adatelemekhez való rendelését ebben az esetben már nem a programozó irányítja, hanem ez a keretrendszer feladata. Amennyiben a feldolgozandó mátrix mérete meghaladná a fizikai végrehajtóegységek számát, akkor a keretrendszer automatikusan több lépésben végzi el a feldolgozást, ilyenkor egymást követően többször fog lefutni a megadott kernel a mátrix különböző részein. Ennek pontos módját szintén a keretrendszer határozza meg, nincs szükség külön programozói munkára (teljes mátrix felbontása kisebb részekre, egyes részekhez szálak rendelése, kernel többszöri elindítása stb.), sőt ennek befolyásolására lehetőség sincs.



2. ábra: Mátrix szorzás mérési eredmények [3]

A technikai háttér részletezése nélkül a 2. ábra mutatja a mérési eredményeket. A feladat jelen esetben két 2048 x 2048 méretű mátrix összeszorzása, amelyek egyszeres pontosságú lebegőpontos számokat tartalmaznak. A mérés során a különböző architektúrákon le lett futtatva ugyanaz a programkód, ezzel meg lett határozva a futásidő. Ez, illetve a végrehajtott lépések darabszáma alapján már egyszerűen kiszámítható a másodpercenként végrehajtott lebegőpontos műveletek száma.

Érdemes megjegyezni, hogy az időmérés során figyelembe lett véve a videokártyák esetén felmerülő adatátviteli idő is. A jelenleg elérhető eszközök ugyanis rendelkeznek azzal a fontos megszorítással, hogy a feldolgozó egységek csak a videokártya saját memóriájához férnek hozzá közvetlenül. A mátrix szorzás tehát valójában ezekből a részlépésekből áll: (1) a két bemeneti és egy kimeneti mátrix számára memória lefoglalása, majd az első kettő adatainak átmásolása a videokártya memóriájába (2) a két bemenő mátrix összeszorzása, az eredmények eltárolása a videokártya memóriában (3) az eredmény mátrix átmásolása a videokártya memóriából a számítógép központi memóriájába. A CPU-k esetében ez a másolás szükségtelen, hiszen a hagyományos CPU már eleve a központi memóriában dolgozik (a memória lefoglalás itt is megjelenik, de ennek időszükséglete elhanyagolható). Ha szigorúan csak a szorzásra koncentrálunk a mérések során, akkor természetesen jogos lehet eltekinteni ettől a késlekedéstől, de ha a gyakorlat szempontjából reálisan szeretnénk összehasonlítani a különböző eszközöket, akkor mindenképpen érdemes figyelembe venni ezt, a csak a GPGPU-k esetén jelentkező hátrányt (tulajdonképpen ezzel egy elemi lépésnek tekintjük az adatok elérését, a számítás végrehajtását, majd az eredmények elérhetővé tételét).

Az első 5 oszlop GPGPU-k teljesítményét mutatja a bemutatott mátrix szorzási művelet során, a következő 6 oszlop pedig néhány jelenleg is elérhető CPU teljesítményét. Jól látható a különbség a két architektúra között, a tesztben szereplő leggyorsabb grafikus kártya teljesítménye 137,18 GFLOPS, míg a hagyományos processzorok között a leggyorsabb is csak 0,83 GFLOPS teljesítményt mutatott.

A táblázat tartalmazza az eszközökben található végrehajtó egységek számát (CPU-k esetében a magok száma, GPU-k esetében pedig az ezeknél jóval egyszerűbb feldolgozóegységek száma). Jól látható, hogy a magok számának növekedésével az elérhető teljesítmény is egyenesen arányosan növekszik. Ennél a feladatnál (és a hozzá tartozó párhuzamosított megoldásnál) tehát tényleg jól kihasználható a végrehajtó egységek nagy száma, a mérési eredmények azt sejtetik, hogy az egységek számának további növelésével nem csak a papíron számítható csúcskapacitás növekszik majd, hanem a gyakorlatban kihasználható teljesítmény is. Ez egyben azt is jelenti, hogy ezek az algoritmusok lineárisan gyorsíthatók további eszközök felhasználásával, ami a gyakorlatban egy meglehetősen jó alternatívát biztosít az információbiztonság gyakorlati feladatai esetén (pl. egy hasonlóan jól párhuzamosítható algoritmussal egy egyébként 1 évig tartó kódtörés 20 videokártyával már csak két hetet vesz igénybe).

N-TEST SZIMULÁCIÓ VIZSGÁLATA

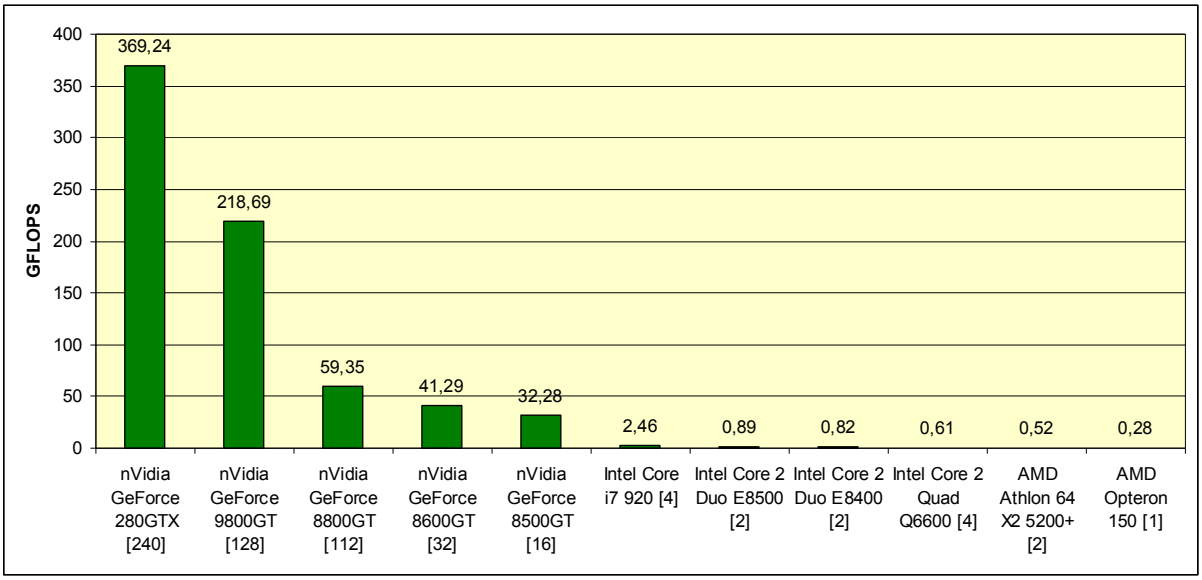
A mátrix szorzás esetében jól látható a párhuzamosítás adta előny, azonban az elért teljesítmény messze nem érte el az elvi maximumot. Ennek legfőbb oka az, hogy figyelembevettük a mátrixok adatainak mozgatását is, ami jelentősen lerontotta a GPU által nyújtott eredményeket.

Célszerű lehet emiatt egy olyan vizsgálat elvégzése, amely szintén jól párhuzamosítható feladatot jelent, azonban meglehetősen kicsi az adatszükséglete (mind a bemeneti, mind pedig a kimeneti oldalon). Erre egy jó választás lehet az N-test szimuláció: eltároljuk N darab test adatait (pozíció, sebesség, tömeg), majd minden egyes iterációs lépésben kiszámoljuk a testek által egymásra ható gravitációs erőket, és ennek megfelelően változtatjuk az egyes testek sebességét/pozícióját.

A testek számának növelésével gyorsan növekszik a szükséges lebegőpontos műveletek száma, mivel minden test esetén ki kell számolni az összes többi test által rá ható erőket, tehát a számításigény $O(N^2)$ nagyságrendű. A memóriában azonban nincs szükség sok adatra, pusztán az egyes testek paramétereit kell eltárolni, ennek szükséglete értelemszerűen N konstansszorosa.

A feladat szintén jól párhuzamosítható, hiszen a számítás legnagyobb része az egyes testek egymásra ható erőinek (majd ezek eredőjének) kiszámítására vonatkozik, ezek pedig egymástól függetlenül elvégezhetők. Joggal várhatjuk tehát, hogy a mérési eredmények a mátrix szorzáshoz viszonyítva még jobban kidomborítják majd a GPGPU-k előnyeit, hiszen itt is nagy párhuzamos számítási kapacitásra van szükség, az adatmozgatás pedig ehhez viszonyítva meglehetősen kevés (a testek kezdőállapota a bemenet, majd pedig a szükséges iterációs ciklusok végén a testek végső állapota a kimenet).

A mérési eredményeket a 3. ábrán látható táblázat foglalja össze:



3. ábra: Mátrix szorzás mérési eredmények [3]

Láthatóan sikerült még jobban megközelíteni az elvi maximális teljesítményt. Szintén jól látható, hogy a végrehajtó egységek számának növelésével szinte egyenes arányban növekszik az elérhető teljesítmény, tehát ez a probléma (illetve a rá kidolgozott megoldás) szintén jól kihasználja a párhuzamos architektúra adta lehetőségeket.

KIEGYENSÚLYOZOTT BINÁRIS-FA FELDOLGOZÁS VIZSÁLATA

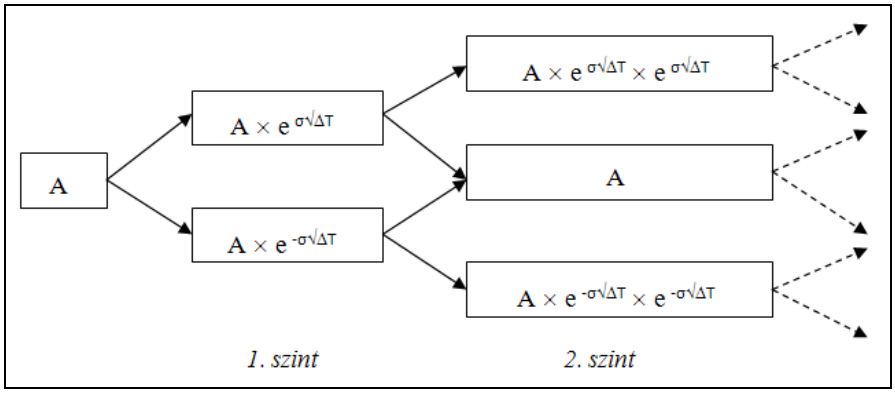
Gyakorlati feladatok során azonban meglehetősen ritka, hogy a teljes probléma megoldása során egymástól független úton lehet számítani az egyes részeredményeket. Általában az egyes részeredmények kiszámítása egymással valamilyen kapcsolatban álló számításokat igényel, vagy esetleg szükség van néhány további részeredmény már előzőleg meghatározott értékére.

Az ilyen számítások egy jelentős része jól modellezhető egy fa adatszerkezet segítségével. Az adatszerkezet levél elemei (a 4. ábrán a legjobboldalibb elemek) tartalmazzák a kiinduló adatokat, amelyekkel azonnal meg lehet kezdeni a szükséges számításokat. A fa felsőbb szintjein lévő elemek kiszámításához azonban már szükség van az alattuk lévő szintek teljes feldolgozására.

Értelemszerűen a teljes párhuzamosítás itt nem valósítható meg, hiszen nem lehet egy időben számolni a fa összes csomópontjához tartozó értéket, pl. a fa gyökérelemének feldolgozása csak az összes többi elemé után következhet. Ez felvett számos kérdést, amelyeket csak jóval összetettebb algoritmusokkal lehet megvalósítani, ezek közül általában a legnagyobb problémát a különféle szinkronizációs lehetőségek beépítése jelenti.

Bár nem lehet minden elemet egy ütemben kiszámítani, egyes részfeladatok még lehetnek meglehetősen jól párhuzamosíthatók. Jelen esetben az adott pillanatban kiszámítható csomópontok jól körülhatárolhatók, ezek mindig a fa levélelemei. Szintén könnyen belátható, hogy (lásd 4. ábra) jelen esetben ezek tulajdonképpen egymástól függetlenül kiértékelhetők.

Egy végrehajtó egység esetén ez egy ciklust jelent, amely végigszalad az összes levélelemen, egyesével kiszámítva a hozzájuk tartozó értékeket. GPGPU-k használata esetén itt is célszerű kihasználni a meglehetősen nagy számú mag által nyújtott lehetőségeket, emiatt minden lépésben érdemes az aktuális levélelemek számának megfelelő mennyiségű szálat indítani, amelyek mind kiszámítják egy-egy levélelem értékét. Miután ez megtörtént, az eddig egyel magasabb szinten lévő csomópontok lesznek az új levél elemek, így már ezekre is végre lehet hajtani ugyanezt a lépést. Értelemszerűen egyre kevesebb levéllel találkozunk, míg végül feldolgozásra kerül a fa gyökére is, ami megadja a végső eredményt.



4. ábra: Az implementált bináris fa felépítése

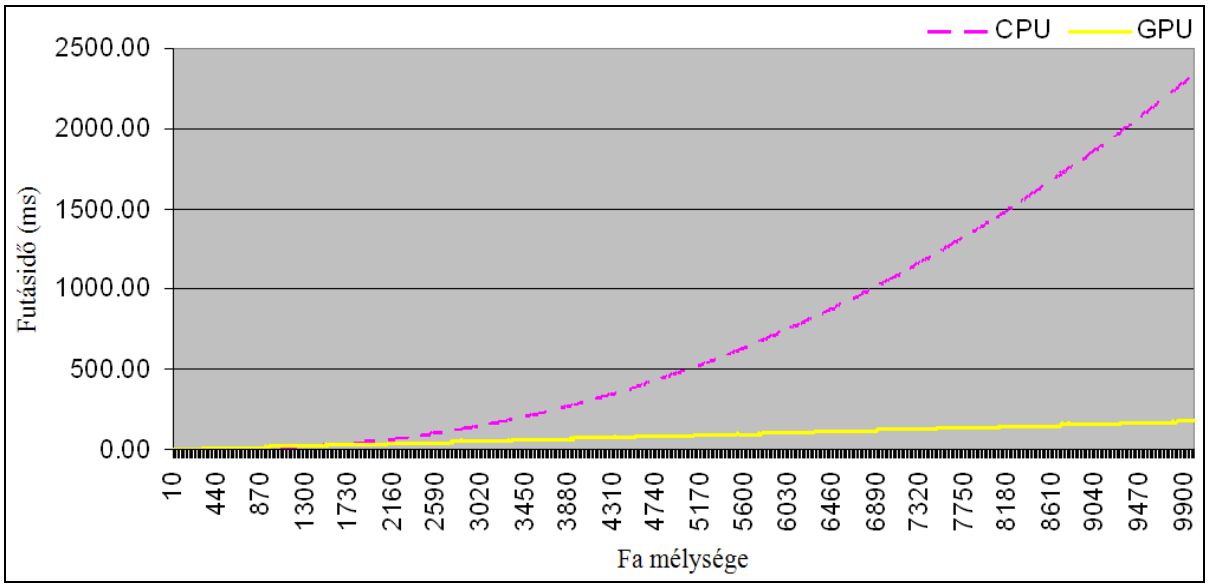
Az implementált program egy egyszerű európai opcióárazást hajt végre, amely esetben a fa gyökérpontja jelenti az aktuális (ismert) árfolyamot, az ebből kiinduló fa egyes ágai jelentik az árfolyamok lehetséges mozgását különféle valószínűségi változóknak megfelelően, a fa csomópontjai az árfolyamok lehetséges értékeit az egyes jövőbeni időpillanatokban, a fa levelei pedig a lehetséges árfolyam értékeket az opció lejáratának időpontjában. A felépítéséhez szükséges az aktuális árfolyam (A), a teljes vizsgált időtartam (T), az iterációk száma (N), az egy iteráció által lefedett időtartam ($\Delta T = T / N$), illetve a volatilitás (σ). A felépített fa levélelemeiben az így kiszámított lehetséges árfolyamértékek alapján már egyszerűen kiszámítható az opció jövőbeni értéke, ez alapján pedig jelenértékszámításokkal (figyelembe véve, hogy a fa egyes ágaira milyen valószínűséggel juthattunk el) egyszerűen kiszámítható az előző szintek csomópontjaiban az opció értéke. Ezt folyamatosan megismételve, a gyökérpontig jutva megkapjuk az opció jelenlegi értékét.

A jelenértékszámításhoz használt GPU kernel kódja:

optionpricing.cu

```
for(int i = N; i > 0; i--)
{
      for(int j = 0; j < i; j++)
      {
            BASE_TYPE upv = stock_tree[j];
            BASE_TYPE downv = stock_tree[j + 1];
            BASE_TYPE prev_value = upv / Mu;
            BASE_TYPE Pu = (prev_value * exprft - downv) /
                           (upv - downv);
            BASE_TYPE option = (Pu * option_tree[j] + (1 - Pu) *
                               option_tree[j + 1]) / exprft;
            option_tree[j] = option;
            stock_tree[j] = prev_value;
      }
}
```

A feladat ideális a GPU-n való végrehajtásra, mivel az adatmozgatások száma minimális, nem szükséges a teljes fa másolására a videokártya memóriája és a központi memória között, elég a fa felépítéséhez szükséges paraméterek átadása (milyen valószínűséggel kell mozogni felfele/lefele, milyen mértékű változások várhatóak, illetve milyen mélységű legyen a teljes fa). A feladat az előzőekben megismert módon, szintenként párhuzamosítva végrehajtható, majd a gyökérelem feldolgozását követően a végeredmény előáll. Ezt természetesen vissza kell juttatni a központi memóriába, de mivel nincs szükség a számítási részeredményekre, ez is csak egyetlen szám másolását jelenti.



5. ábra: A kiegyensúlyozott bináris fa feldolgozásának futásideje

A nagyfokú párhuzamosság és a minimális memóriaátvitel miatt jelentősen kisebb végrehajtási időre számíthatunk a GPGPU használatával, mint a CPU esetében, emiatt jelen esetben nem is a különböző eszközök összehasonlítása az érdekes, sokkal inkább az, hogy a különböző mélységű fák feldolgozása során hogyan viszonyul egymáshoz a két különböző architektúra által nyújtott teljesítmény. Ennek megfelelően az 5. ábra a végrehajtási időt mutatja a fa méretének függvényében (a konkrét mérések során GPU: nVidia GeForce

280GTX, 240 végrehajtó egység; CPU: Intel Core2 Quad Q6600 2,4Ghz, 4 mag; a rendszer 32 bites lebegőpontos számokkal számolt mindkét esetben).

Az eredményeket célszerű a fa mélységének (és így a végrehajtott iterációk számának) függvényében vizsgálni:

- Kis számú iteráció esetében (N < ~1500): bár a GPU jóval több maggal rendelkezik, mint a CPU, így magát a tényleges számítást már itt is gyorsabban el tudja végezni, mint a hagyományos eszközök, azonban ez az előny nem használható ki jól, mivel a program futtatása során meglehetősen sok járulékos költség keletkezik (kernel indítása, kernel leállítása, szinkronizáció, bemenő adatok átvitele, eredmények kiolvasása a videokártya memóriából).

- Nagy számú iteráció esetében (N > ~1500): az előbb említett járulékos költségek természetesen továbbra is fennállnak, azonban a GPU nagy számítási kapacitásával már tudja ezeket ellensúlyozni. A tesztelt eszköz 240 végrehajtó egységgel rendelkezik, felépítéséből adódóan ezeket is csak akkor tudja megfelelően hatékonyan kihasználni, ha az általa futtatandó szálak száma jóval meghaladja ezt a számot. Emiatt amikor már nagy számú levelet kell feldolgozni, akkor a GPU teljes kihasználtsággal működhet, ez pedig jelentős előnyt jelent számára a CPU hagyományos 4 magjával szemben.

Tehát megadott iterációs szám alatt a GPU használata felesleges, egy megadott (gyakran csak a gyakorlati tesztek során jól meghatározható) határ felett azonban kimondottan jól teljesítenek ezek az eszközök. Érdemes azonban megjegyezni, hogy a feladat többi tulajdonsága is hozzájárult ehhez az eredményhez: kevés memóriaátvitelre volt szükség, nagy mennyiségű lebegőpontos számítást kellett végrehajtani, ezek a számítások pedig jelentős részt egymástól függetlenek voltak.

EREDMÉNYEK ÉRTÉKELÉSE

A mérési eredményeket figyelembe véve tehát egyértelműen kimondható, hogy a GPGPU technológia által nyújtott teljesítmény nem csak elméleti lehetőségeket rejt, hanem jelentős részt a gyakorlatban is jól kihasználható. Mindig az adott alkalmazás dönti el, hogy érdemes-e az új technológiát felhasználni, ez ugyanis csak akkor vezet jó eredményre, ha meglehetősen nagy számú párhuzamos műveletet kell elvégezni, enélkül a hagyományos CPU kevés, ámde jóval fejlettebb magjai egyszerűbb és hatékonyabb megoldást kínálnak. Nehéz lenne pontosan definiálni, hogy melyek ezek a feladatok, néhány egyszerű tapasztalati alapelvet azonban célszerű figyelembe venni:

- *Memória átvitel minimalizálása:* a GPGPU architektúrák egyik legnagyobb hátránya jelenleg, hogy ezek az eszközök csak a saját memóriájukban képesek műveletvégzésre, tehát minden bemenő adatot ide kell másolni, illetve a további feldolgozás részére az eredményeket innen vissza kell másolni a központi memóriába. Ez meglehetősen nagy időveszteséget jelent (a mátrix szorzás példájában közel 90%át az eltel időnek), emiatt a nagy memóriaigénnyel rendelkező alkalmazások esetén érdemes figyelembe venni ezt a veszteséget. Amennyiben ez lehetséges, célszerű a kimenet/bemenet mennyiségét csökkenteni, pl. ha a bemenet kevesebb paraméter alapján egyszerűen generálható, célszerű azt a videokártyán belül elvégezni.

- *Adatpárhuzamos végrehajtás:* az eszközök felépítéséből adódóan a nagy számú végrehajtó egység csak ugyanazon programkód futtatására alkalmas, az egyes különálló aritmetikai egységek ugyanis nem feltétlenül rendelkeznek saját vezérléssel. Emiatt bár az elágazások és ciklusok lefordíthatók és futtathatók ezeken az architektúrákon is, azok indokolatlan használata jelentősen csökkenti a teljesítményt. Ezen esetekben célszerű a programkódot átstrukturálni, amennyiben ez nem lehetséges, felülvizsgálni, hogy érdemes-e GPGPU segítségével megvalósítani a feladatot.

- *Nagy mennyiségű szál futtatása:* szintén a vizsgált eszközök felépítéséből adódik, hogy igazán csak akkor használhatók hatékonyan, amennyiben meglehetősen nagy mennyiségű szálat kell párhuzamosan futtatniuk. Ez az alsó határ nem feltétlenül csak a végrehajtó egységek számát jelenti (ha ennél kevesebb szálat kell indítani, akkor teljesen nyilvánvaló, hogy nem használjuk ki az eszköz adta lehetőségeket), hanem annak többszöröse is lehet, mivel a vizsgált kártya működése során alkalmas arra, hogy ha egy szál várakozó állapotba kerül (pl. memóriából adatra vár) akkor az ezt futtató végrehajtó egységet ez idő alatt egy másik szál futtatására használja fel. Mindez az átütemezés hardveresen, többlet veszteség nélkül történik.

- *Optimalizálás szükségessége:* természetesen a hagyományos CPU alapú programozás során is jelentős az optimalizálás szerepe, azonban az új technológia esetén ez még lényegesebb lehet. Egyrészt a fordítóprogramok még nem kellőképpen kiforrottak, így az optimalizálás jelentős részt a programfejlesztő feladata, másrészt az egyes eszközök egymástól jelentősen különböznek, így a konkrét futtatást végző hardverre testreszabott kód (memóriaméretek, memóriakialakítás stb.) nagyságrendekkel jobb eredményeket nyújthat.

A felmerülő hátrányoktól eltekintve azonban jól látható, hogy az információbiztonság területén lényeges aritmetikai számításokat (mind a kódolás/dekódolás, mind pedig az esetleges kódolt szövegek feltörése esetén) a hagyományos CPU-khoz képest nagyon nagy teljesítménnyel hajtják végre ezek a piacon napjainkban is elérhető eszközök.

Felhasznált irodalom

[1] Nvidia CUDA Compute Unified Device Architecture Programming Guide, Version 2.0, Nvidia, June 2008

[2] GeForce 280GTX spec.: http://www.nvidia.com/object/product_geforce_gtx_280_us.html (Letöltés ideje: 2009.07.01)

[3] Cs. Csillingh, Cs. Iványi, S. Szénási: Multiplatform Performance Analysis of Parallel Workloads on Graphics Hardware, Morgan Stanley-BME Financial Innovation Centre Kick-off & Workshop, Budapst, 2009

[4] Nvidia CUDA Compute Unified Device Architecture Programming Guide, Version 1.1, Nvidia, Nov. 2007