Acta Agraria Kaposváriensis (2010) Vol 14 No 3, 213-222
Kaposvári Egyetem, Állattudományi Kar, Kaposvár
Kaposvár University, Faculty of Animal Science, Kaposvár

Képfeldolgozási algoritmusok gyorsítása párhuzamos feldolgozás segítségével

Ureczky B.
Budapesti Műszaki és Gazdaságtudományi Egyetem, Méréstechnika és Információs Rendszerek Tanszék
1117 Budapest, Magyar tudósok körútja 2.

ÖSZEFOGLALÁS

A mellkasfelvételek egyik legzavaróbb jellegzetessége a csontok árnyékának jelenléte, ami a tüdő elemzése során csak hátráltató tényező és akár egy komoly elváltozás észrevételének esélyét is csökkentheti. A technika fejlődésével ma már nagy felbontású, jó minőségű digitális képek készülnek, amik alkalmasak számítógépes feldolgozásra és lehetőség nyílik számítógép segítette diagnózis, ún. CAD rendszerek kialakítására, melynek során az orvosokat segítő funkciók valósíthatóak meg, mint például elváltozások keresése, vagy csontárnyékok eltűntetése. Ezen képfeldolgozó algoritmusok általános hátránya, hogy lassúak, ami csökkenti az alkalmazhatóságát annak nagyszerűsége ellenére, hiszen CAD-rendszereket csak real-time környezetre érdemes fejleszteni, ahol az orvos gombnyomásra el tudja tűntetni a csontárnyékokat; ha erre perceket kellene várnia, akkor nem tudja használni. A grafikus processzorok hihetetlen (és egyre nagyobb) számítási képességének kihasználása egy feltörekvőben levő, folyamatosan fejlődő technológia. Munkámban ismertetem az NVIDIA® CUDA™ rendszerét és annak alkalmazhatóságát, hogy hogyan lehet párhuzamos feldolgozás segítségével algoritmusokat gyorsítani, mint például egy csontárnyék-eltűntető eljárást.
(Kulcsszavak: párhuzamos programozás, GPGPU, CUDA, képfeldolgozás)

ABSTRACT

Accelerate Image Processing Algorithms using parallelized processing

B. Ureczky
Budapest University of Technology and Economics, Department of Mesurement and Information Systems
H-1117 Budapest, Magyar tudósok körútja 2.

One of the most embarrassing feature of the lung X-ray images is the shadows of the bones. It could be disturbing at the medical examination if it is the lung which is being examinated, not the bones. These shadows can hide abnormalities and can decrease the chance of detecting them. During the evolution of the technology, it is available nowadays to take digital, high-resolution pictures of the lung, which can be processed by computers in order to make Computer Aided Diagnose (CAD) systems which help the radiologists with new features like detecting abnormalities or eliminate bone shadows. The disadvantages of these algorithms are that they are too slow to be used in a real-time application. A bone shadow eliminating function can be also useful for a abnormality detector program in order to increase its efficiency but it is unuseless for the doctor if he has to wait for minutes to get the result. The application of the graphics cards with tremendously high computing capabilities is an up-to-date, persistently

developing technology. In my presentation, I introduce the NVIDIA® CUDA™ system, and how to make an algorithm faster with using parallel computations.
(Keywords: parallel programming, GPGPU, CUDA, image processing)

BEVEZETÉS

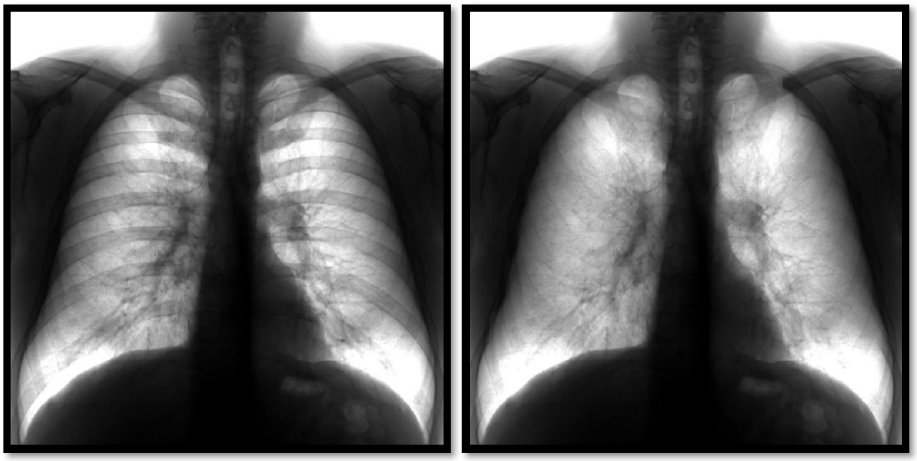
Magyarországon nagy szerepe van a tüdőszűrésnek tekintettel a nagyszámú daganatos megbetegedésre és egyéb elváltozásokra. A technika fejlődésével ma már alacsony sugárterhelés mellett készíthetünk jó minőségű digitális röntgenképeket, melyek alkalmasak a leletezésre. Ugyan léteznek ennél modernebb, precízebb vizsgálatot lehetővé tevő térbeli képet készítő gépek, ám ezek sugárterhelésük és áruk miatt nem alkalmazható szűrővizsgálatként.

A BME-MIT tanszék részt vesz egy projektben, ami a meglévő infrastruktúra jelentős fejlesztése nélkül hoz létre egy orvosi döntéstámogató CAD rendszert, melynek segítségével megkönnyíthető az orvosok leletezése. Egy röntgenkép alapján számos hasznos vizsgálatot, kiértékelést végezhetünk számítógéppel, melyek az orvosok munkáját segítik. (*Ureczky,* 2009)

Az algoritmusok és az orvosok szempontjából is hasznos funkció, ha a mellkas-felvételekről eltüntetjük a zavaró csontárnyékokat (*1. ábra*). Egyetlen felvétel alapján ez igen nehéz feladat, de nem kivitelezhetetlen. Erre született már megoldás, de lassú, másodpercekben, illetve percekben mérhető futásidővel, ami a gyakorlatban nem használható, hiszen az orvos csak úgy tud használni egy funkciót, ha az azonnal lefut, és nem kell az eredményre várnia, ezért kiemelkedő fontosságú a meglévő algoritmusok gyorsítása.

1. ábra

Csontárnyék eltűntetés



Forrás (*Source*): *Horváth* (2009)

Figure 1: Eliminating the shadows of the bones.

Az utóbbi években előretörtek a grafikus kártyák hihetetlen számítási kapacitásuk révén, ami az új CUDA technológia segítségével viszonylag könnyen kiaknázható.

ANYAG ÉS MÓDSZER

A CUDA™ technológia

2006 novemberében az Nvidia bemutatta a CUDA™ architektúrát, ami egy eszköz-független (bár jelenleg csak az NVIDIA saját grafikus kártyái támogatják), általános programozási modell párhuzamos programok írására (*Nvidia*, 2009).

Szálhierarchia

A módszer egyik alapgondolata a *szálhierarchia*. Fontos, hogy a feladat jól *dekomponálható* legyen, azaz fel tudjuk bontani olyan elemi eljárásokra, amit sokszor kell elvégeznünk. Ez lehet például bizonyos elemek összeadása, vagy ezeken bármilyen művelet végzése. Ezt az eljárást, függvényt hívjuk *kernel*nek, a függvényt végrehajtó programszálakat pedig *szál*aknak (thread). Ezt a kernelt nem egymás után futtatjuk sokszor, mint a soros programozás során, hanem egyszerre, ugyanazzal a paraméterezéssel, nagyon sok kis egységen, egy-egy ún. *szálprocesszor*on (TP, Thread Processor). Azonban nem szükséges egzaktul megfogalmazni, hogy melyik TP-n milyen melyik szál fusson, mi csupán elindítunk nagyon sok szálat egyszerre és az eszköz osztja szét a feladatokat a TP-k között. A szálakat hierarchiába rendezve *blokk*okra (block) csoportosíthatjuk, amik egy *rács*ot (grid) alkotnak. Végül ezt a rácsot adjuk át az eszköznek, az pedig ütemezi a *multiprocesszor*ai (MP, Multiprocessor) között a blokkokat. A multiprocesszor több szálprocesszorral rendelkezik, ezek között ütemezi a szálakat (*2. ábra*).

2. ábra

Szálhierarchia: szál – blokk – rács

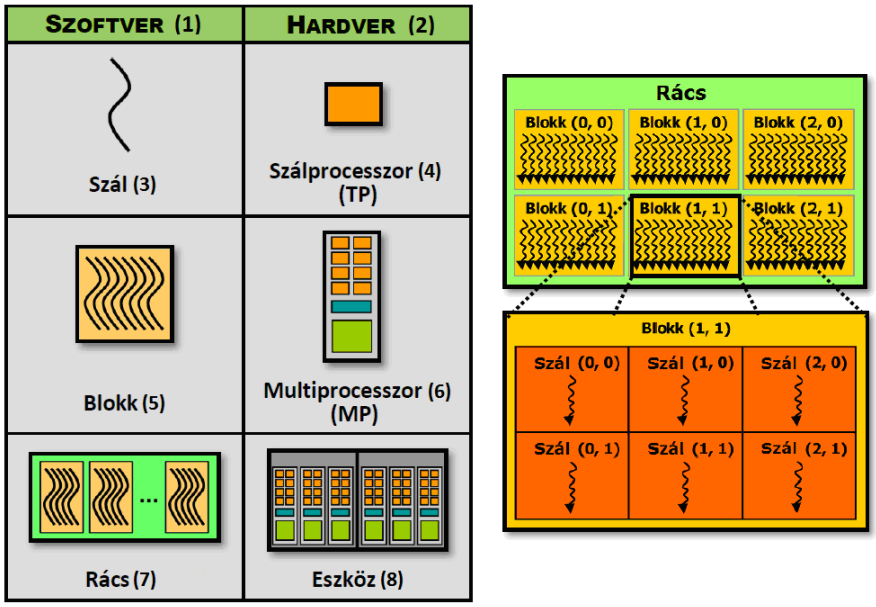


Figure 2: Thread hierarchy: thread – block – grid

Software(1), Hardware(2), Thread(3), Thread Processor(4), Block(5), Multiprocessor(6), Grid(7), Device(8)

Ami elsőre nem világos, hogy miben tér el a szálak futása, ha ugyanazzal a paraméterezéssel indítjuk őket és ugyanazt a függvényt futtatják. A szálak rendelkeznek azon információval, hogy ők éppen milyen indexelésűek (ami lehet 1, 2, vagy 3 dimenziós is) a blokkban, illetve a blokk milyen indexelésű a rácsban. Ez felfogható egy rejtett paraméterként, ami már lehetőséget ad arra, hogy különböző feladatokat végezzenek és a memóriának különböző részein (tehát más adatokon) tudjanak számolni. Például egy képfeldolgozás esetén egy-egy szál megfeleltethet magának 1 pixelt, vagy egy régiót a képen és annak környezetében csinálhat valamit.

A hierarchiának köszönhetően jól skálázható programokat írhatunk, mindezt átlátszóan, tehát a programozás során nem szükséges azzal foglalkoznunk, hogy milyen eszköz futtatja a programunkat: Ha egy 512 CUDA maggal rendelkező eszközön futtatjuk, akkor is ugyanazt a rácsot adjuk át az neki, mintha az 16 CUDA maggal rendelkezne, csak ez utóbbi esetben a futási idő 32-szerese lenne (*3. ábra*). Akár 1 magon is futtathatjuk, lényegében ez történik emulációs módban a CPU-n.

Az új Fermi architektúrában bevezetésre kerülő konkurens kernelfuttatás lehetősége pedig tovább optimalizálja a terheléselosztást a több rács egyidejű ütemezése révén (*4. ábra*).

3. ábra

Skálázhatóság: Rács futtatása egy 2, illetve egy 4 multiprocesszoros eszközön

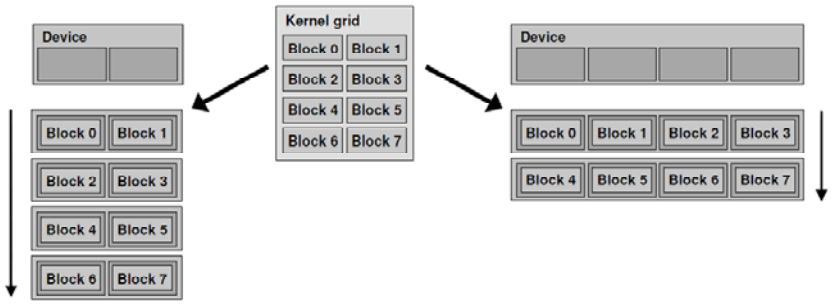
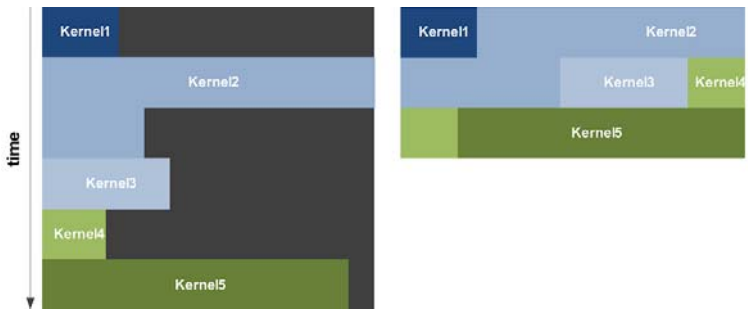


Figure 3: Scaleability: Execute a grid on a device with 2 or 4 multiprocessors.

4. ábra

Soros és konkurens kernelfuttatás



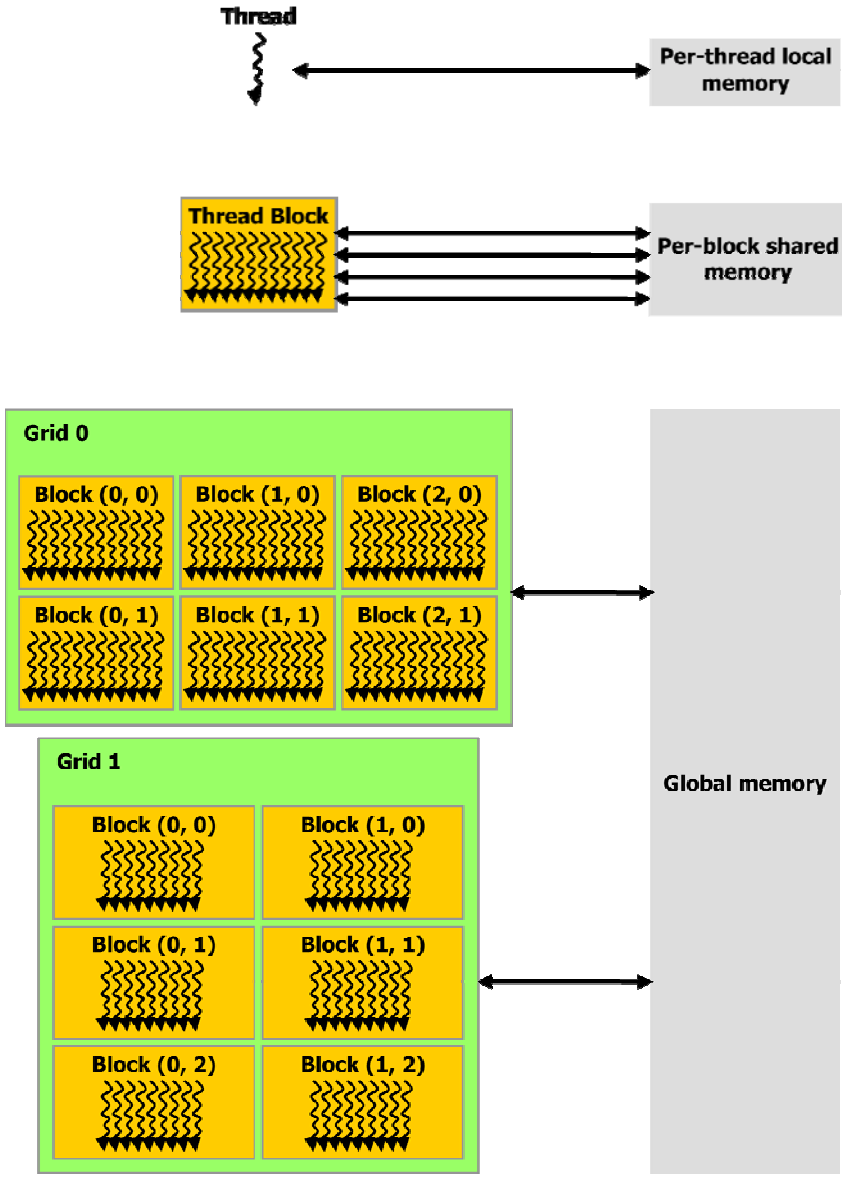
Forrás (*Source*): *Nvidia*, (2009)

Figure 4: Parallel and concurrent execution of the kernel.

Memóriahierarchia
A hierarchia egy további jellegzetessége, hogy a különböző hierarchia-szinten levő elemek különböző kapcsolatban állnak egymással (*5. ábra*).

5. ábra

Memóriahierarchia: lokális, osztott és globális memória



Forrás (*Source*): *Nvidia*, (2009)

Figure 5: Memory hierarchy: local, shared and global memory

Egy szál az őt futtató szálprocesszor *lokális memóriá*ját (local memory) használja, ezt más nem látja, csak ő, viszont nagyon gyors az írás és olvasás, akár a CPU-ban a regiszterek használata. Ha egy másik – de az ő blokkján belüli – szállal akar együttműködni, akkor ezt megteheti az őket futtató multiprocesszor *osztott memóriá*jában (shared memory).

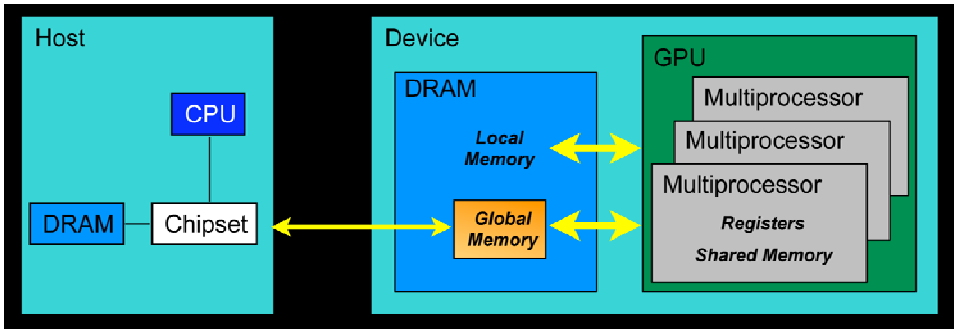
Ha egy olyan szállal akar együttműködni, ami nem az ő blokkjában található, akkor ezt sajnos csak a még lassabb, *globális memóriá*ba (global memory) való írással-olvasással teheti meg.

A rácsok közötti kommunikáció a globális memórián keresztül lehetséges, ám a jelenlegi (Fermi előtti) változatokban még nem futtathatóak párhuzamosan a rácsok, csak egymás után, sorosan.

Ezen kívül még fontos megemlíteni, hogy a GPU globális memóriája nem azonos a CPU-éval, a szálak nem látják a CPU memóriáját, hasonlóan a CPU sem férhet hozzá a GPU multiprocesszorainak és processzorainak osztott és lokális memóriájához. Így egy kernel indítása előtt a szükséges adatokat a GPU globális memóriájába kell töltenünk, ahonnan a szálak kiolvashatják és esetlegesen áttölthetik a lokális vagy osztott memóriába, majd a végeredményt vissza kell tölteniük a globális memóriába ahonnan a CPU az eredményt áttölti a saját memóriájába és csak innen használhatja az eredményt (*6. ábra*).

6. ábra

Kommunikáció a hoszt és eszköz között



Forrás (*Source*): *Nvidia*, (2009)

Figure 6: Communication between the host and the device

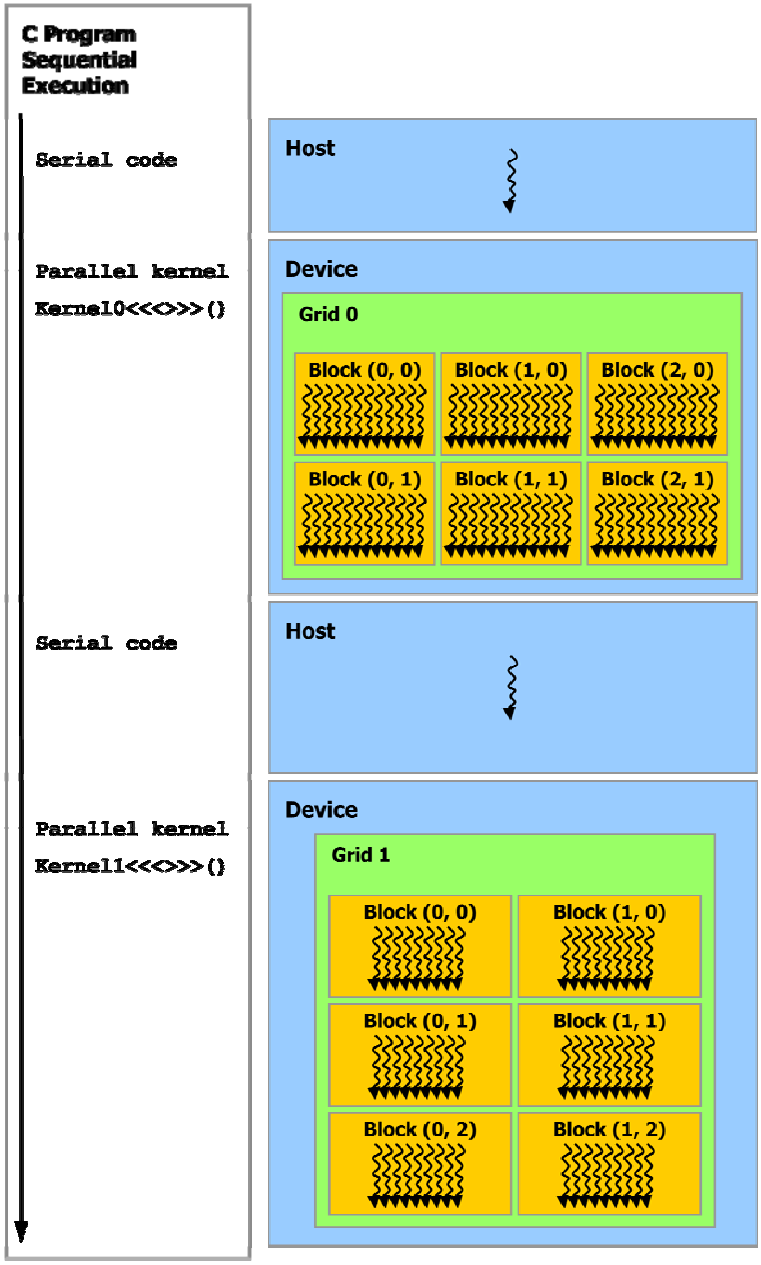
Heterogén programozás

Összességében tehát egy heterogén programozási modellel állunk szemben, külön kell programozni a CPU-t, ami soros kód futtatására képes, és külön a GPU-t, ami párhuzamos feldolgozásra képes. A GPU-n a kernelt mindig a soros kód indítja el, ami ettől kezdve párhuzamosan futhat a soros kód mellett (*7. ábra*). Ez egy újabb párhuzamosításra ad lehetőséget, hiszen egyszerre végezhet számításokat a CPU és a GPU.

Ez alól kivételt tesz a memória másolása, ami közben a GPU nyilván nem végezhet számításokat, a CPU-nak meg kell várnia, amíg befejezi azt, csak akkor kezdheti el az adatmozgatást. Viszont a memóriamozgatást nem a CPU maga végzi, ahogyan az előző ábrán látható, hanem a Chipset, így a memóriamozgatás közben a CPU ezzel párhuzamosan is futtathat soros kódot.

7. ábra

Heterogén programozás: A soros kód a hoszton fut, a párhuzamos az eszközön



Forrás (*Source*): *Nvidia*, (2009)

Figure 7: Heterogenous programming: the serial code is executed on the host, while the parallel code is executed on the device.

Alkalmazás

A párhuzamos feldolgozást kihasználva a futásidőt töredékére csökkenthetjük. Ehhez szükséges, hogy sok, hasonló adaton kelljen ugyanolyan műveleteket végezni. Nagyszerű példa erre a mátrixokkal számolás, ahol egy szál általában egy mező értékét számolja, vagy egy képfeldolgozó eljárás, ahol egy szál általában egy pixel értékét számolja. A gyakran használt különböző szűrések, átskálázás, Fourier-transzformált előállítása nagyszerű példa. Ezen kívül sok más helyen alkalmazható még, például sok adaton végzett asszociatív művelet esetén. Egy egyszerű maximumszámításhoz nem szükséges *n* lépés, hiszen elegendő processzor esetén szálanként egy pár maximumát képezve $log_2(n)$ lépés alatt meghatározható a maximum (*8. ábra*).

8. ábra

Maximumkeresés párhuzamos feldolgozással

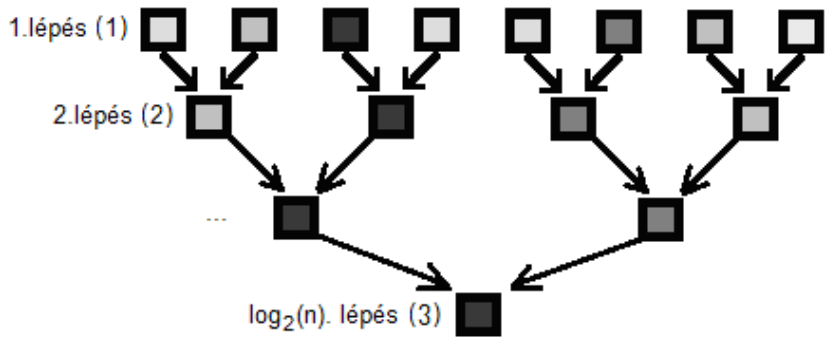


Figure 8: Finding the minimum using parallel computation

1^{st} step(1), 2^{nd} step(2), …, $log_2(n)^{th}$ step(3)

Ennél a példánál persze nem teljes a magkihasználtság, a jól dekomponálható számítások esetén a magok számának növelésével közel fordított arányban csökken a futási idő, így egy több száz magos GPU esetén a futásidő akár többszázad részére csökkenhet (*Nvidia*, 2009).

EREDMÉNY ÉS ÉRTÉKELÉS

A továbbiakban egy saját példán mutatom be az elérhető sebességnövekedést. (*Ureczky*, 2009). Egy borda- és kulcscsont-eltűntető algoritmusának élfinomító eljárását implementáltam párhuzamos környezetben (*1. táblázat*). Ez többször is meghívódik egy futás során. Az adatok egy mellkas-felvétel csontozatának meghatározása közben készült az eredeti és a módosított program elemzésével, ahol a finomító eljárás többször is meghívódik. A párhuzamos algoritmus futásidejét a dll-ben lévő függvényen belül és a külső c# programból való hívás során is mértem, ezeket mutatják a nettó és bruttó oszlopok. Az adatok egy mellkas jobb oldali felére vonatkoznak, a teljes algoritmus futásideje ennek kétszerese. Az adatok több egymás utáni futás eredményének átlaga.

A mért adatok a notebookomban lévő, 32 CUDA maggal rendelkező Nvidia Geforce GT130-as kártyával készültek. Mint látható az elért sebességnövekedés kb.30-

szoros, ám a dll hívása során fellépő járulékos időveszteségek miatt csak ötszörös. Az elért eredményen még lehet javítani, például ha csökkentjük a két környezet közötti kommunikációt. Egyik lehetőség erre, hogy kevesebb paramétert adunk át a dll-nek (Bizonyos tömbök átadása helyett elegendő lenne ezek generáláshoz szükséges adatok), a másik pedig, hogy át kellene strukturálni az egész C# programot úgy, hogy több finomítás is hívható legyen egyetlen utasítással. Ez megtehető, mert a bordák külön-külön számolhatóak, így csak az iterációk számával megegyező számú hívás lenne szükséges.

1. táblázat

Futási eredmények ezredmásodpercekben

	Soros algoritmus (1)	**Párhuzamos algoritmus (2)**	
		nettó (3)	**bruttó (4)**
Kulcscsont (3 függvény hívás) (5)	1845	43	140
	7718	72	172
	1053	26	117
∑:	10 616	141	429
Bordák (48 függvény hívás) (6)	161	9	96
	161	8	112
	250	11	101
	…	…	…
	697	22	122
	995	30	124
	1019	30	116
∑:	21 871	784	5 424
Összes futási idő (7):	32 487	926	5 853
	~32 mp	~1 mp	~6 mp

Table 1: Runtimes in milliseconds

Serial algorithm(1), Parallel algorithm(2), Net(3), Gross(4), Collarbone, calling 3 functions(5), Ribs, calling 48 functions(6), Total runtime(7)

KÖVETKEZTETÉSEK

Mint látható, az eredmények bíztatóak, ha a probléma megfelelően dekomponálható és párhuzamosítható. Ez általában igaz a képfeldolgozó algoritmusokra, de nem is szükséges az egész program átírása, csupán a kritikusabb, különösen időigényes részeké. Egy viszonylag olcsón (pár tízezer forintért) beszerezhető, PC-be beépíthető, többszáz maggal rendelkező GPU-val a futásidő akár több század részére csökkenthető, így nem véletlen, hogy manapság egyre felkapottabb a terület, egyre több program használja ki a GPU-k segítségével történő gyorsítást. Agrárinformatikában használt digitális képfeldolgozási algoritmusok során is hasznos lehet a technológia bevezetése, ahol kritikus a válaszidő, főként real-time rendszerekben.

IRODALOM

Horváth Á. (2009): Mellkasröntgen felvételek elemzése: bordarendszer feltérképezése, Diplomamunka, BME, Budapest, 6-10. p., 41-53. p.

Nvidia (2009): CUDA Programming Guide, Version 2.3. [Online] http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf

Ureczky B. (2009): Mellkasröntgen felvételeken csontárnyékok keresésének hatékony implementálása, Szakdolgozat, BME, Budapest, 11-16. p., 27-32. p., 37-38. p.

Levelezési cím (*Corresponding author*):

Ureczky Bálint
H-7400, Kaposvár, Kálvária u.83/a
Tel: +36-70-276-0564
email: ubalint@gmail.com