CUDA

Optimalizálási kérdések

- Magas szintű optimalizálás
  - Soros kód párhuzamosítása
  - Mennyi a várható teljesítmény növekedés?
    - Erős skálázódás (Amdahl törvény)
      - Mennyire lineáris a skálázódás a párhuzamosítás növelésével?
      - $S = \frac{1}{(1-P) + \frac{P}{N}}$

        S: maximális teljesítmény növekedés

        P: párhuzamosítható kód aránya a soros kódban

        N: processzorok száma
      - GPGPU példa: N: nagyon nagy

        $S = 1/(1-P)$, ha a párhuzamosítható a ¾-e a kódnak

        $S = 1/(1-3/4) = 4$

Optimalizálási kérdések

- Magas szintű optimalizálás
  - Mennyi a várható teljesítmény növekedés?
    - Gyenge skálázódás (Gustafson törvény)
    - Mennyire lineáris a skálázódás
      - A processzorok számának növelésével
      - Fix probléma méret processzoronként
      - Azaz a problématér a processzorok számával nő

$$S = N + (1 - P)(1 - N)$$

Optimalizálási kérdések

- Hogyan párhuzamosítsunk?
  - Segédkönyvtárak használatával
    - Meglévő soros kód párhuzamosítása
    - A meglévő program már használ CPU-n futó könyvtárat
  - Párhuzamosító fordító
    - A teljesen automatikus vektorizáció nehéz probléma
    - Praktikusan a fordítót segítő direktívákat kell használni
      - OpenMP – CPU vektorizáció
      - OpenACC – GPU vektorizáció
  - Saját GPU kód

Optimalizálási kérdések

- Numerikus pontosság és sebesség
  - A GPU erősen érzékeny az adat típusra
    - Float vagy Double?
    - Gyakran más eredményt adnak a számítások
    - A Double legalább kétszer lassabb
    - A lebegő pontos műveletek nem asszociatívak
    - IEEE 754-nek megfelel, néhány kivétellel
  - A CPU-hoz képest más a számítások pontossága
    - A CPU és GPU más reprezentációt használ
    - CPU 80bit, GPU 64bit

Optimalizálási kérdések

- Milyen elméleti mérőszámok vannak?
  - Sávszélesség
    - Elméleti sávszélesség a grafikus kártya specifikációjából
      - Tesla M2090: GDDR5 RAM, 1.85GHz, 384bit széles busz
      - Az elméleti sávszélesség: $(1.85 \cdot 10^9 \cdot (384/8) \cdot 2)/10^9 = 177.6 GB/s$
      - ECC használatakor a sávszélesség 20%-al kisebb
    - Effektív sávszélesség a mérésből
      - Memória írások és olvasások száma az eltelt idő alatt
      - Effektív sávszélesség: $((B_r + B_w)/10^9)/\text{time}$
      - Példa: 2048x2048-as mátrix másolása

$$((2048^2 \cdot 4 \cdot 2)/10^9)/\text{time}$$

Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Adatcsere a host és device között
    - Az elméleti sávszélesség kicsi (8GB/s)
      - Minimalizáljuk az adatátvitelt
    - Az átvitelnek is van költsége
      - Kevés nagyobb blokk a sok kicsi helyett
    - Speciális memória szervezés
      - Pinned memória
      - Write-combining memória
      - Zero-copy memória
      - Drága és korlátos erőforrások

Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Adatcsere a host és device között
    - Másolás és kernel futtatás átlapolása
      - A host és device kód átlapolása

```
cudaMemcpyAsync(a_d, a_h, size, cudaMemcpyHostToDevice, 0);
kernel<<<grid,block>>>(a_d);
cpuFunction();
```

      - Memória másolás és kernel futtatás átlapolása

```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
cudaMemcpyAsync(a_d, a_h, size,
                cudaMemcpyHostToDevice, stream1);
kernel<<<grid,block,0,stream2>>>(b_d);
```

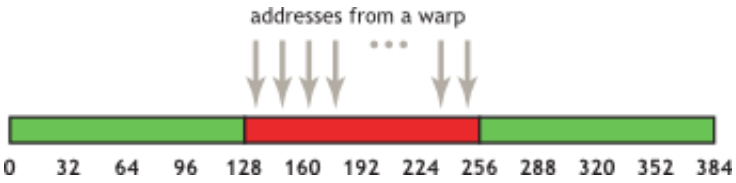
Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Adatcsere a host és device között
    - Problématér darabolása

```
size = N * sizeof(float)/nStreams;
for(i=0; i<nStreams; ++i)
{
  offset = i*N/nStreams;
  cudaMemcpyAsync(a_d+offset, a_h+offset, size,
                  cudaMemcpyHostToDevice, stream[i]);
  kernel<<<N/(nThreads*nStreams),
           nThreads,0,stream[i]>>>(a_d+offset);
}
```

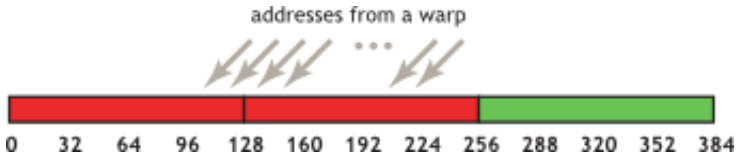
Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - A globális memória elérésének mintája (coalesced access)
    - Egyszerű minta
      - A warp egymás utáni memória címekhez fordul
      - 128bit-es tranzakció

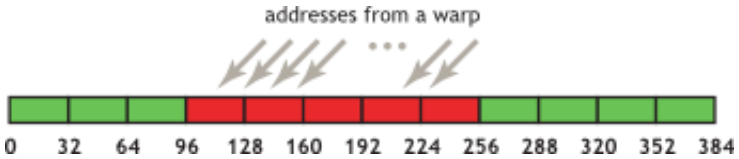


Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - A globális memória elérésének mintája (coalesced access)
    - Egyszerű minta, igazítás nélkül
      - A warp egymás utáni memória címekhez fordul
      - Több tranzakció szükséges
        - L1

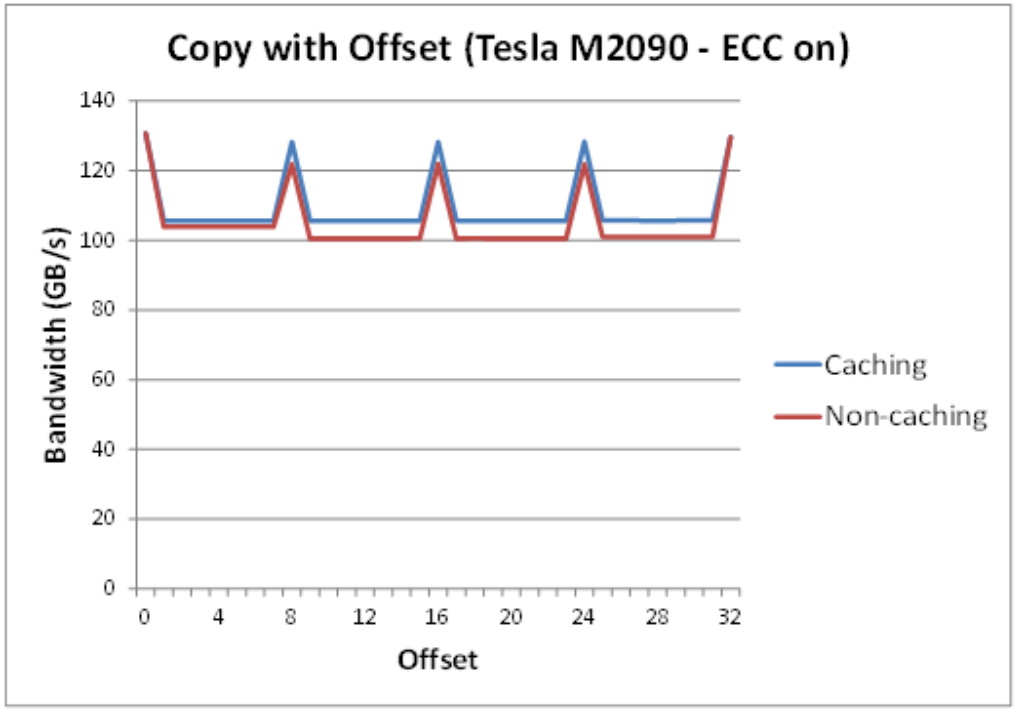


        - L2



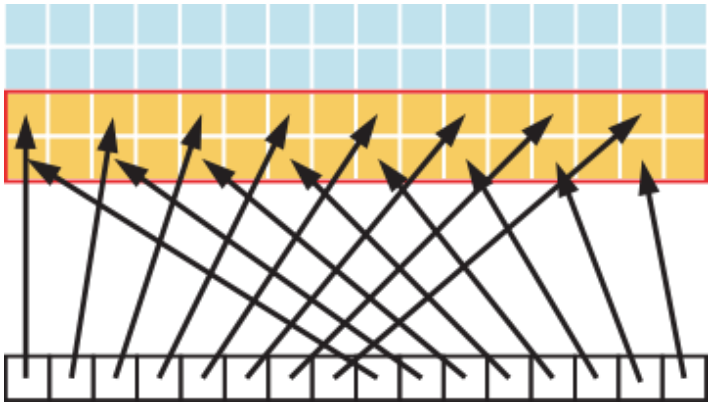
Optimalizálási kérdések

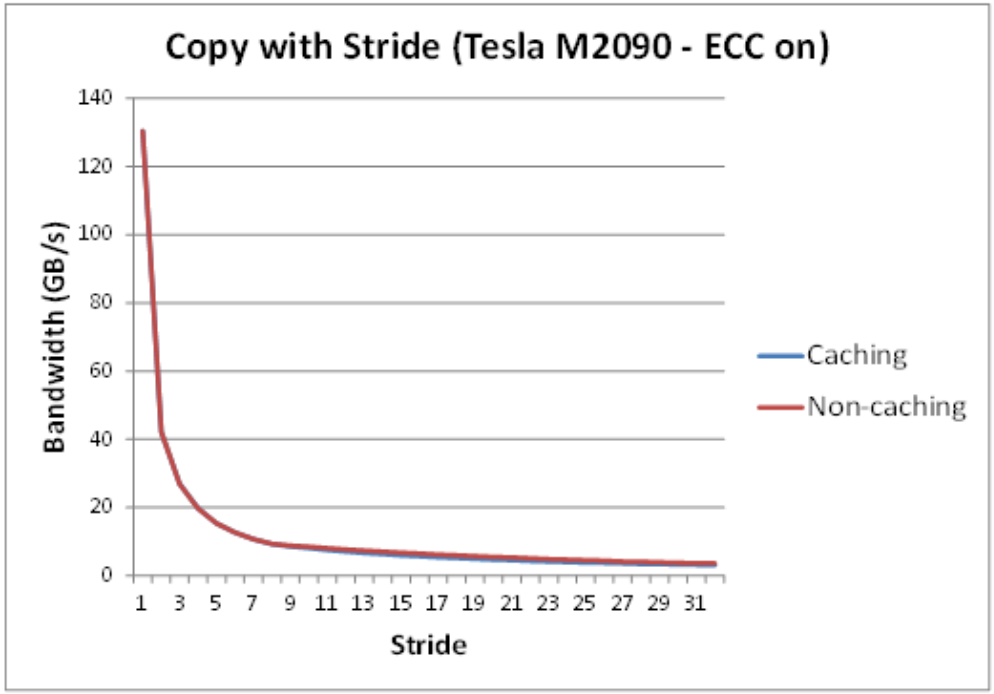
- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - A globális memória elérésének mintája (coalesced access)
    - Az igazítás hatása



Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - A globális memória elérésének mintája (coalesced access)
    - Az elérendő elemek közötti offset hatása





Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - Az osztott memória elérése
      - Az osztott memória több egyforma méretű blokkokra van osztva
      - Az egyes blokkok párhuzamosan elérhetőek
      - Azonos blokkba irányuló kérések sorosítva lesznek
    - CC 1.x
      - Sávszélesség: bankonként 32bit két órajel alatt
      - Egymást követő 32bites szavak egymást követő bankokban
      - Warp méret 32, Bankok száma 16

Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - CC 2.x
      - Sávszélesség: bankonként 32bit két órajel alatt
      - Egymást követő 32bites szavak egymást követő bankokban
      - Warp méret 32, Bankok száma 32
    - CC 3.x
      - Sávszélesség: bankonként 64bit egy órajel alatt
      - Egymást követő 32bites szavak egymást követő bankokban,
        Vagy egymást követő 64bites szavak egymást követő bankokban
      - Warp méret 32, Bankok száma 32

Optimalizálási kérdések

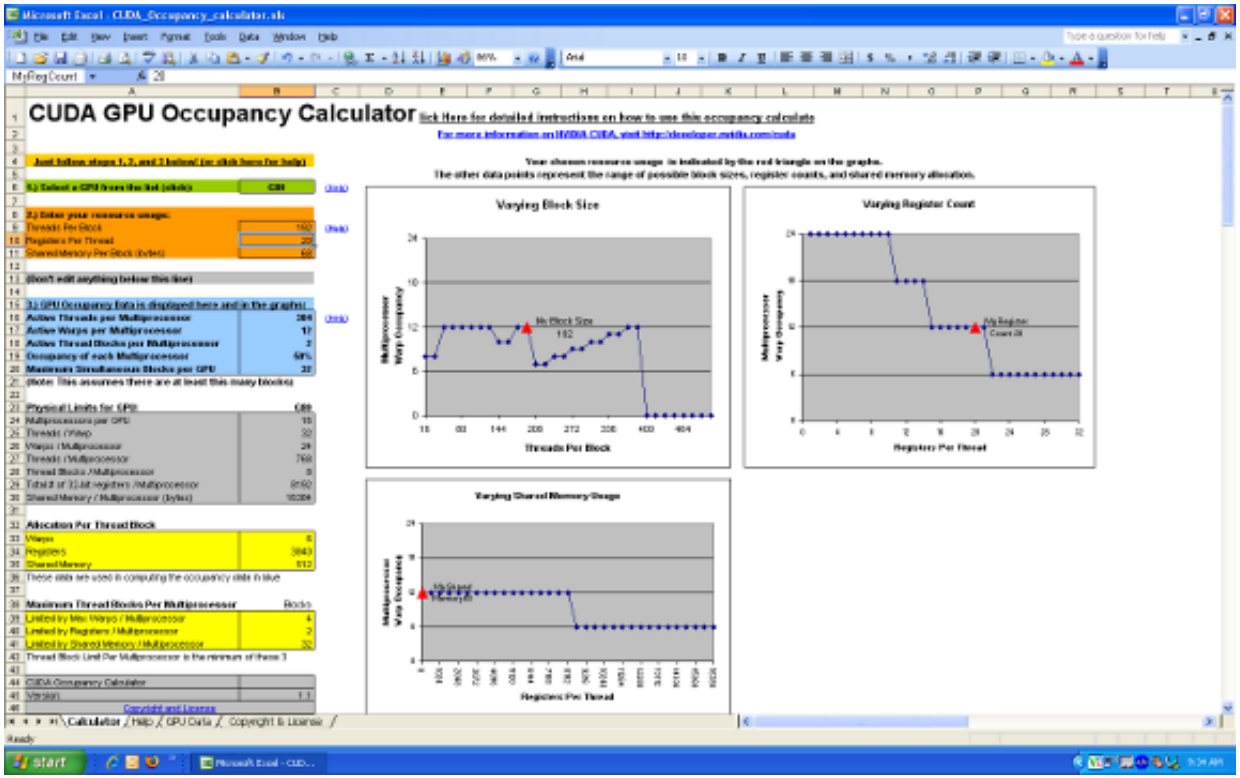
- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - Lokális memória elérése
      - Nem a multiprocesszorban van
      - Automatikusan használja a fordító, ha szükséges
      - Minimalizálni kell a használatát
    - Textúra memória
      - Cache-elt memória terület
      - 2D lokalitásra optimalizálva
      - Konstans késleltetés
        - A cache találat csak a sávszélesség igényt csökkenti

Optimalizálási kérdések

- Hogyan kezeljük a memóriát?
  - Memória elérés GPU kódból
    - Konstans memória elérése
      - 64KB a méretű cache-elt terület
      - A konstans cache sorosítja a kéréseket, ha nem azonos címről van szó
    - Regiszterek
      - Általános esetben szinkron elérésű
      - Read-after-write késleltetés
        - 24 órajel, de megfelelő számú párhuzamos szál esetén amortizálódik
      - Bank ütközés
        - Az ütemező megpróbálja elkerülni
        - Nincs közvetlen ráhatásunk

Optimalizálási kérdések

- Hogyan válasszuk meg a munkaméretet?
  - Occupancy
    - Az aktív és maximálisan indítható warpok aránya



Optimalizálási kérdések

- Hogyan válasszuk meg a munkaméretet?
  - Párhuzamos kernel futtatás
    - Eszközfüggő a párhuzamos kernelek száma
    - Streamek segítségével párhuzamosítható
  - Regiszter függőségek
    - A késleltetés csökkenthető megfelelő számú szál indításával
  - Blokk méret megválasztása
    - Szálak száma a warp méret többszöröse legyen
    - Minimum 64 szál legyen blokkonként
    - 128-256 szál blokkonként jó kiindulás
    - A késleltetés csökkenthető több kisebb blokkal
    - Az osztott memória korlátai

Optimalizálási kérdések

- Aritmetikai pontosság vagy sebesség?
  - Egyszeres vagy dupla pontosság
  - Speciális matematikai függvények
  - Típusok közötti konverzió
  - Intrinsics függvények (__function)
    - Közvetlenül a hardver utasításokra képződnek le
  - Fordítási opciók
    - -ftz=true (denormált számok legyenek nullák)
    - -prec-div=false (osztás pontossága)
    - -prec-sqrt=false (gyök számítás pontossága)

Optimalizálási kérdések

- Kód szervezés
  - Elágazások, divergencia
    - Warpon belüli elágazásokat kerülni kell
    - Célszerű az elágazásokat a warp mérethez igazítani
  - Elágazás előrejelzés
    - Feltételes utasítás folyamok
      - Elágazások feltételes kiterítése (utasítás limit)
      - Speciális kiértékelési folyamat, ha a feltétel nem teljesül
    - #pragma unroll

Optimalizálási kérdések

- Kód szervezés
  - Ciklus változók
    - Előjeles egész az előjel nélküli helyett
  - Elágazást tartalmazó kódban kerüljük a szinkronizálást
    - A __syncthreads() hívásra minden szálnak rá kell futnia
    - Kernelből hívható függvények speciális kezelése
      - Divergens kódból hívás helyett flaggel jelezzünk