WIGNER GPU Lab

(GP)GPU kezdetek

Avagy hogyan lesz az algoritmusból implementáció

Lectures on Modern Scientific Programming
Wigner RCP
23-25 November 2015

D. Berényi – M. F. Nagy-Egri

Tartalom

- Számítógépes grafika
- GPU hardver ismeretek
- GPGPU API történelem
- Miért OpenCL?
- OpenCL C API
- OpenCL C++ API
- Algoritmus könyvtárak

Számítógépes grafika

„Ha ismered az ellenséget és ismered önmagad, nem kell félned száz csatától sem. Ha magadat ismered, de az ellenségedet nem, minden egyes győzelem után egy vereség vár rád. Ha sem magadat, sem az ellenséget nem ismered, minden egyes csatát el fogsz veszíteni."

- Sun Tzu: A háború művészete

Inkrementális kép szintézis

- Leginkább játékokban alkalmazott kép alkotási eljárás
- Célja: egy virtuális, 3 dimenziós színtér elemeit megjeleníteni a 2 dimenziós képernyőn
- A színtér elemeit háromszögekből (polygon) építjük fel
- A háromszögeket koordináta transzformációk sorozatával juttatjuk a képernyőre
- A transzformációk jellege közel azonos objektumról objektumra
- A háromszögeket transzformáció után textúrákkal kiszínezzük

Affin geometria

- A 3 dimenziós Euklideszi-teren ható lineáris transzformációk legtöbbje leírható 3×3-as mátrixokkal
- Affin transzformációnak hívjuk a párhuzamosságot megtartó trafók összességét
- A 3 dimenziós Euklideszi-térhez tartozik egy 4 dimenziós homogén koordinátázott tér amiben minden számunkra lényeges trafó leírható 4×4-es mátrixokkal

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_{14} \\ a_{21} & a_{22} & a_{23} & b_{24} \\ a_{31} & a_{32} & a_{33} & b_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

Csúcspont transzformációk

Modell koordináták

Világ koordináták

Kamera koordináták

Képernyő koordináták

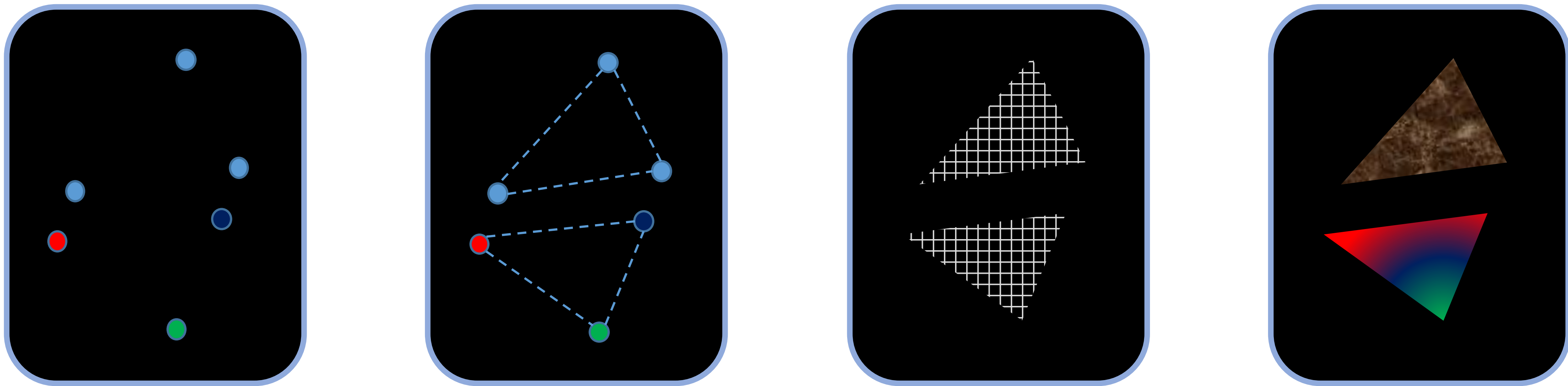
Modell transzformáció

Nézeti transzformáció

Vetítés

Minden egyes transzformációt egy 4x4-es mátrix ír le.

A raszterizálás menete



Primitívek összeszerelése

Raszterizálás

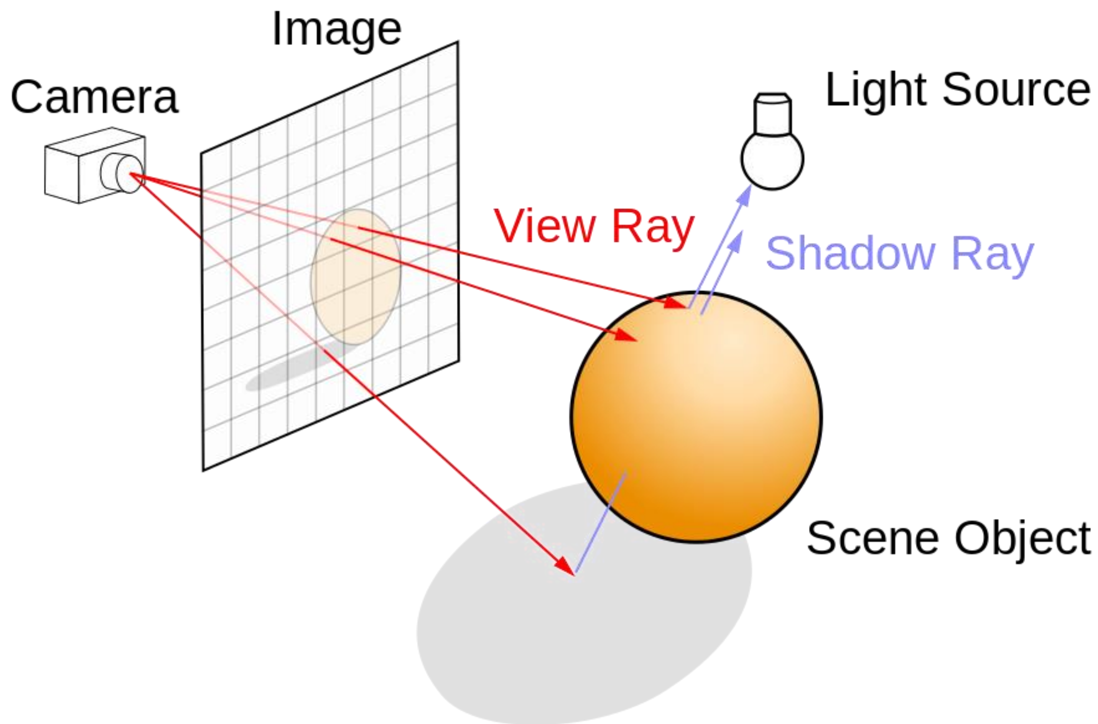
Interpolálás, textúrázás, színezés

Minden egyes lépés fix funkciós elemekkel támogatott.

OpenCL

Sugárvetés, sugárkövetés

Wigner GPU Lab



- Animációs filmek kedvelt képalkotási eljárása
- A fénysugarak útját követi le a fényforrásoktól a képpontokig
  - A hatékonyság miatt pont a fordítottja történik
  - A képpontokból kiindulva keressük a fényforrásokat
- A sugarak felösszegzik a különböző felületkről származó színeket

OpenCL

Renderelési technikák

Inkrementális kép szintézis

- A render egyetlen közelítő megoldása
- Valós időben rajzolható
- Fix funkciós támogatás
  - Erre született

Sugárkövetés

- A render egyenlet pontos megoldása
- Tipikusan offline rajzolható
- Általános számolásnak minősül
  - GPU-n annyi sebből vérzik…

Wigner GPU Lab

GPU hardver ismeretek

Born to RENDER!

D. Berényi – M. F. Nagy-Egri

Mi a különbség?

Wigner GPU Lab

CPU

- Késleltetésre kondicionált hardver
- Általános végrehajtási egység
- Fix utasítás architektúra

GPU

- Áteresztésre kondicionált hardver
- Adatpárhuzamos végrehajtási egység
- Virtuális utasítás architektúra

WIGNER GPU Lab

Graphics Processing Unit (GPU)

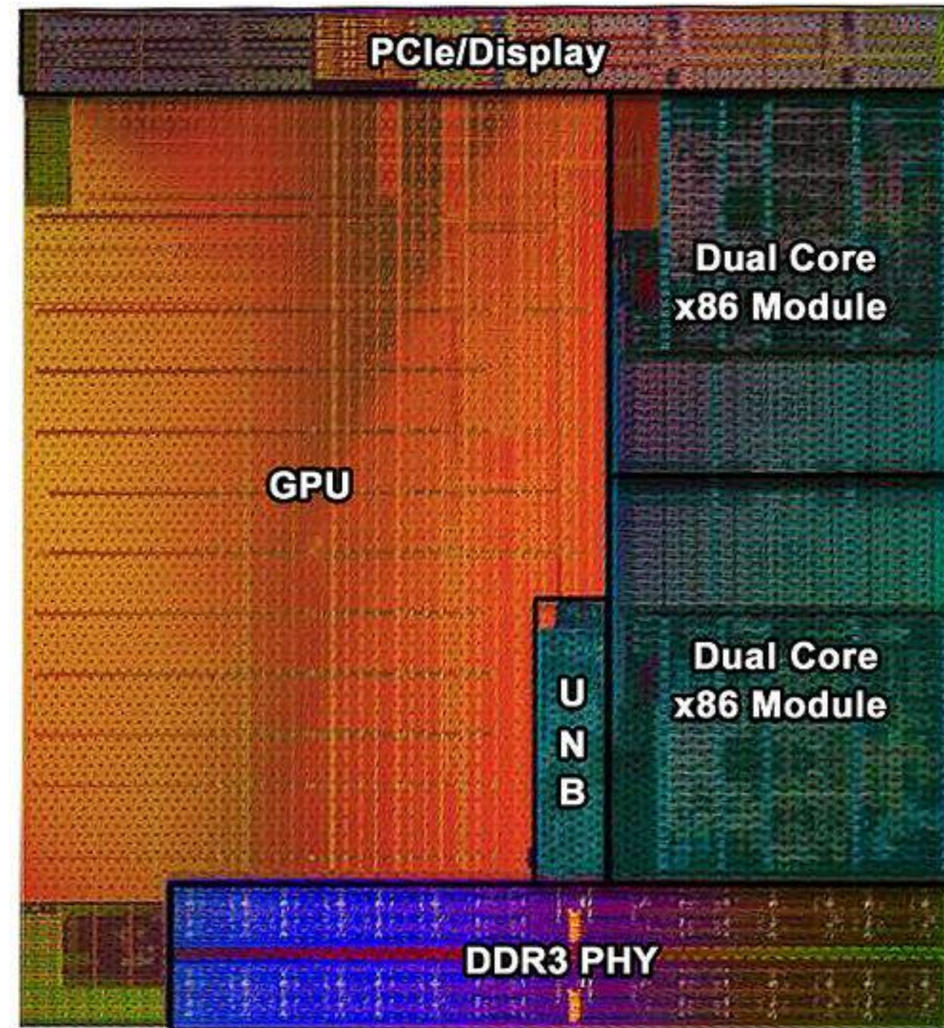
- Rengeteg TOC
- Valamennyi fix funkciós egység
- Adatpárhuzamosság a cél

chipworks

AMD Tahiti (Radeon HD7970) áramköri fotó (kontrasztosítva)

Accelerated Processing Unit (APU)

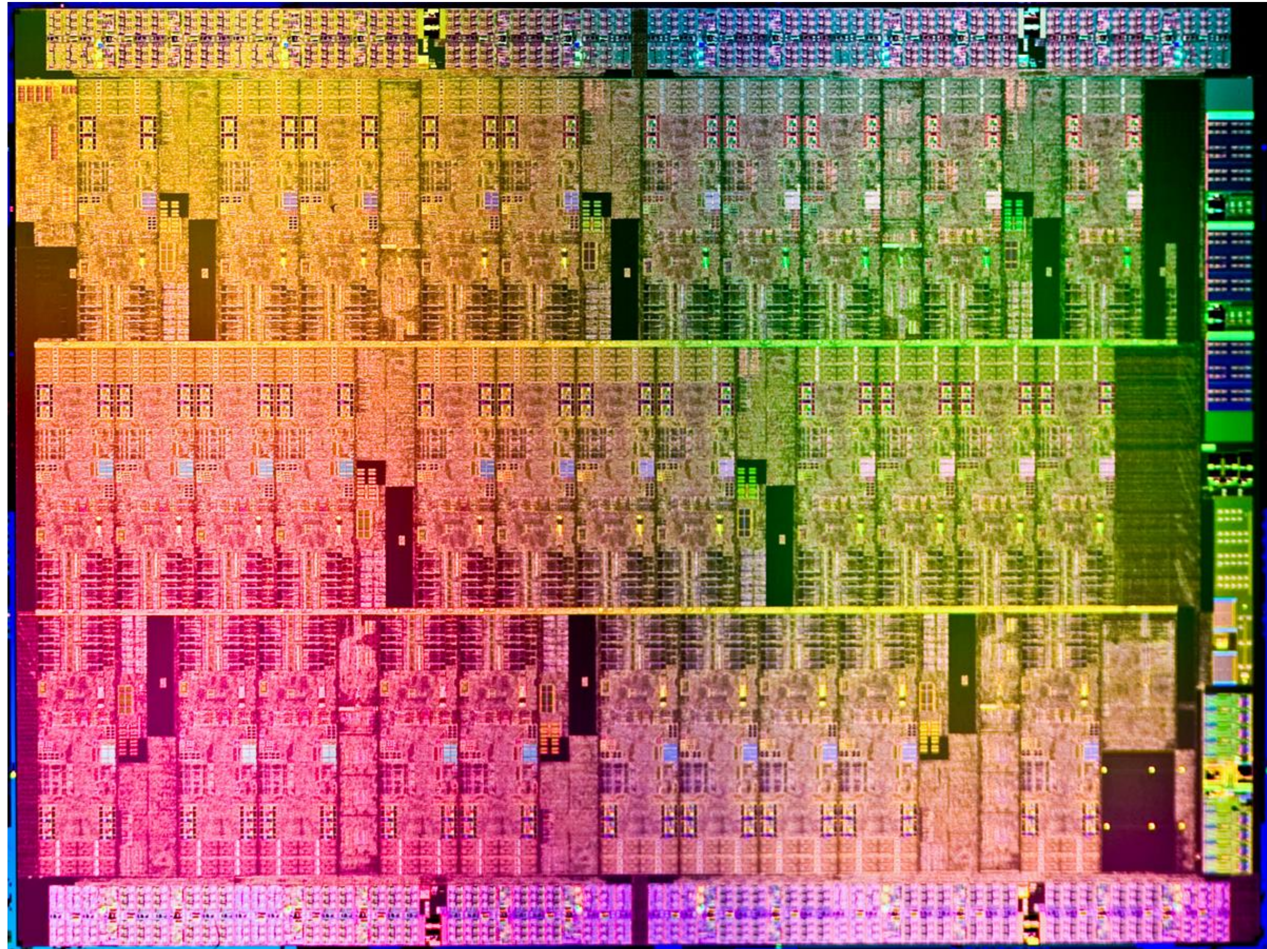
- Kevés LOC
- Rengeteg TOC
- Valamennyi fix funkciós egység
- Egyre inkább ez a tendencia



AMD Kaveri áramköri fotó (színezett)

Many Integrated Cores (MIC)

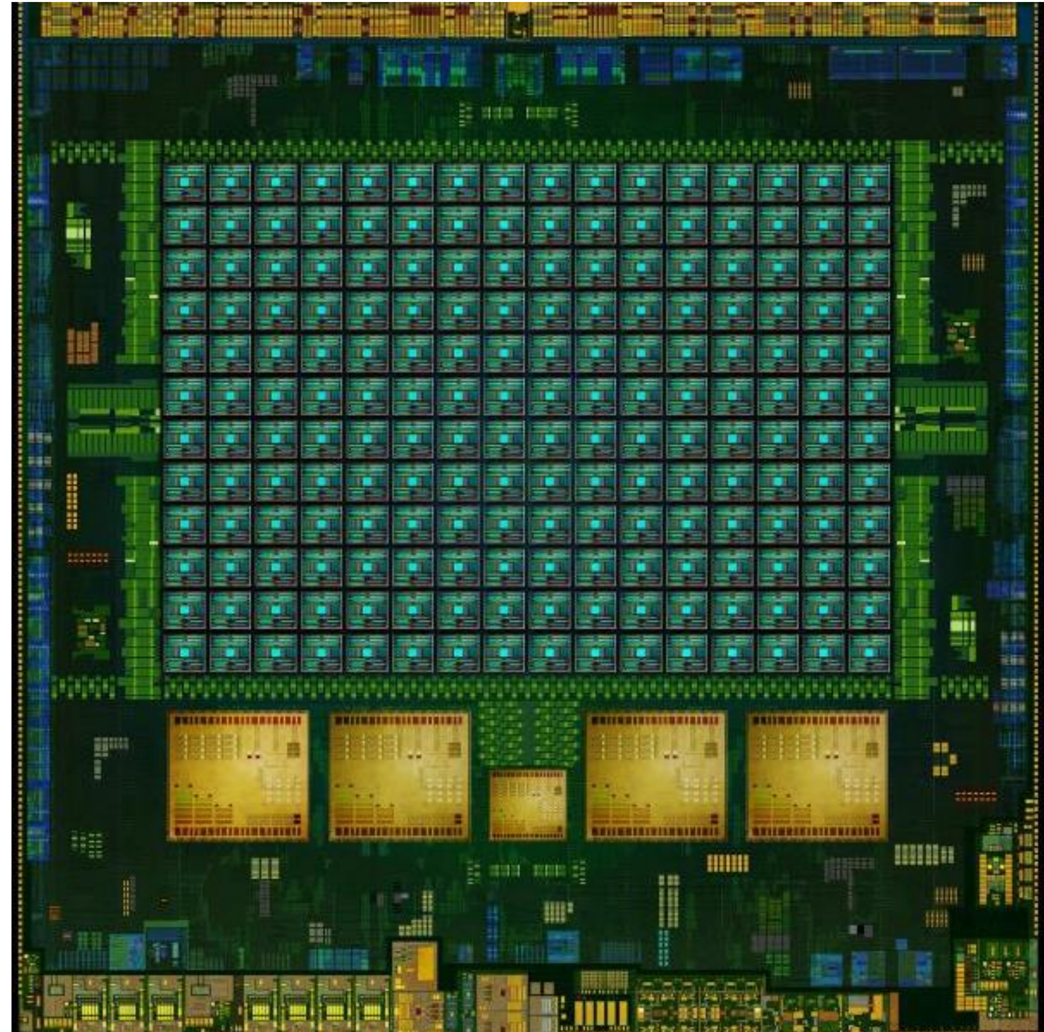
- Project Larrabee néven indult
- Felkiáltás: grafikát x86-tal
- Rengeteg „LOC”
- Egyszerre LOC és TOC
- Igazából se LOC, se TOC



Intel Knight’s Corner áramköri fotó

Mobile computing

- X86 helyett ARM magok
  - CISC kontra RISC
- LOC kevesebb helyet foglal
- Több IGP vagy több fix funkciós
- A média tartalomtól terhes mobil világban ez a legkézenfekvőbb

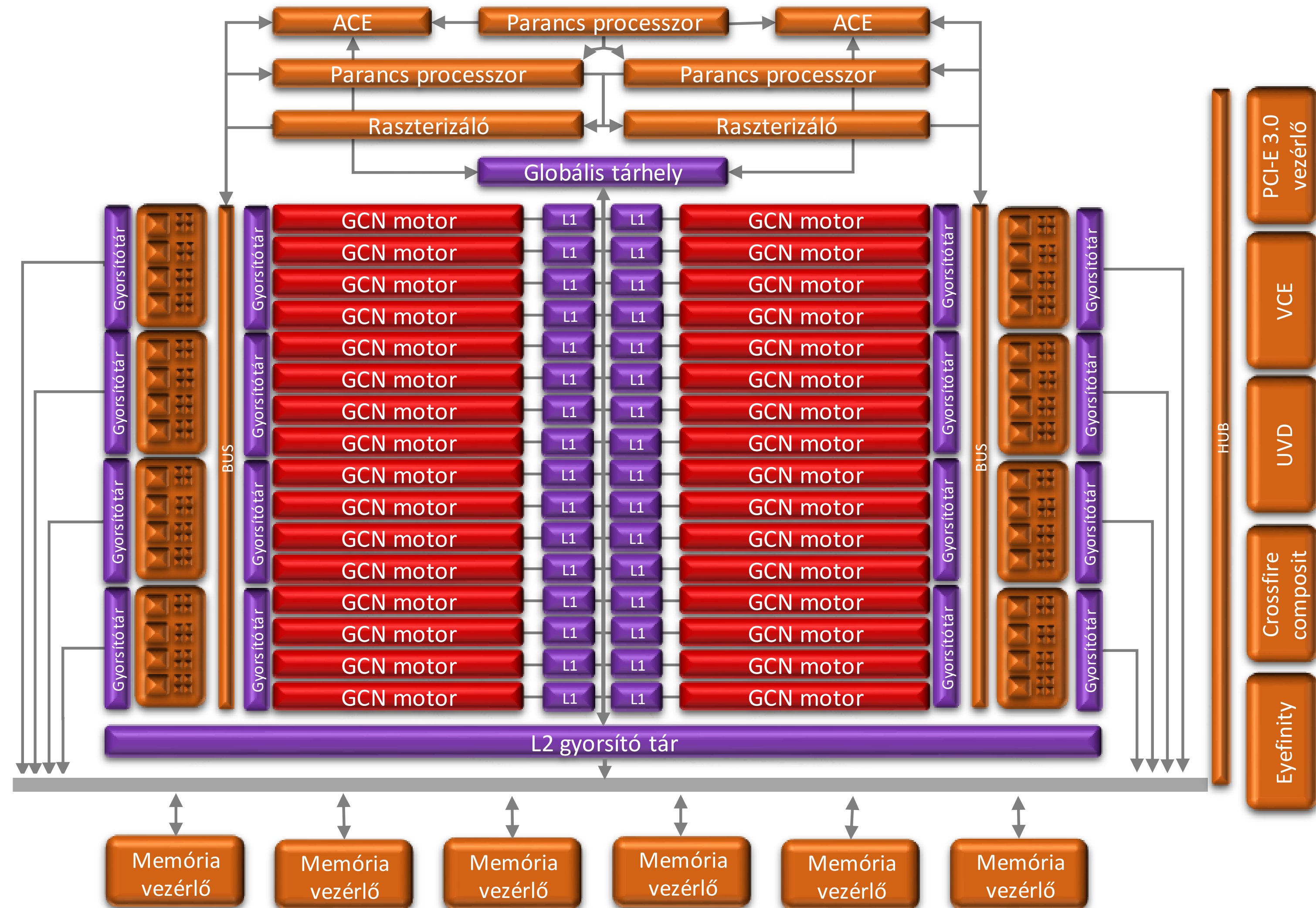


Nvidia Tegra K1 áramköri fotó (színezett)

AMD Graphics Core Next 1.0

WIGNER GPU Lab

- Masszívan párhuzamos architektúra
- Erősen redundáns, ellenálló a gyártási hibákkal szemben
  - Hierarchikus felépítés (cache, ACE+GCN)
- Fix funkciós egységek, nem csak grafikára
  - ACE: Asynchronous Compute Engine
  - GDS: Global Data Share
  - UVD: Universal Video Decode
  - VCE: Video Coding Engine



AMD Tahiti (Radeon HD7970) sematikus ábra

OpenCL

GCN Számoló Egység részletek

WIGNER GPU Lab

- Általános célokra optimalizált számítási egység
  - 1 CU = 4 SIMD motor
  - 1 SIMD = 16 regiszter
  - 1 regiszter = 32 bit
- Regiszter késleltetés (!)
- Közös memória
  - Kvintesszenciája a hatékony GPU használatnak
- Csatolt működés
  - A SIMD egységek minden sávja egy külön szálat fog futtatni
  - 1 SIMD motorhoz azonban 1 program számláló tartozik
  - Nincs igazi elágazás

Csak olvasható adat gyorsítótár (16kB / 4 GCN motor)

Fix funkciós

Utasítás gyorsítótár (32kB / 4 GCN motor)

GCN motor

Utasítás betöltő

Program és elágazás számláló

Program és elágazás számláló

Program és elágazás számláló

Program és elágazás számláló

Utasítás irányító

Elágazás- és üzenet-küldő

Export/GDS dekódoló

Vektor memória dekódoló

Skalár dekódoló

Vektor dekódoló

Gyorsítótár dekódoló

64kB közös memória

Skalár egység

8kB REG

INT ALU

SIMD motor 64kB REG MP ALU

SIMD motor 64kB REG MP ALU

SIMD motor 64kB REG MP ALU

SIMD motor 64kB REG MP ALU

Írható/olvasható L1 gyorsítótár (64kB)

• • •

GCN motor

Utasítás betöltő

Program és elágazás számláló

Program és elágazás számláló

Program és elágazás számláló

Program és elágazás számláló

Utasítás irányító

Elágazás- és üzenet-küldő

Export/GDS dekódoló

Vektor memória dekódoló

Skalár dekódoló

Vektor dekódoló

Gyorsítótár dekódoló

64kB közös memória

Skalár egység

8kB REG

INT ALU

SIMD motor 64kB REG MP ALU

SIMD motor 64kB REG MP ALU

SIMD motor 64kB REG MP ALU

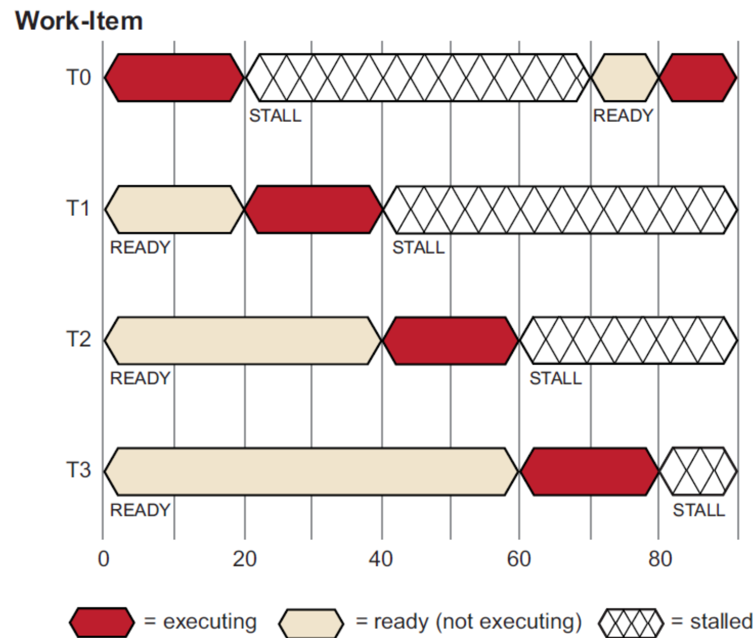
SIMD motor 64kB REG MP ALU

Írható/olvasható L1 gyorsítótár (64kB)

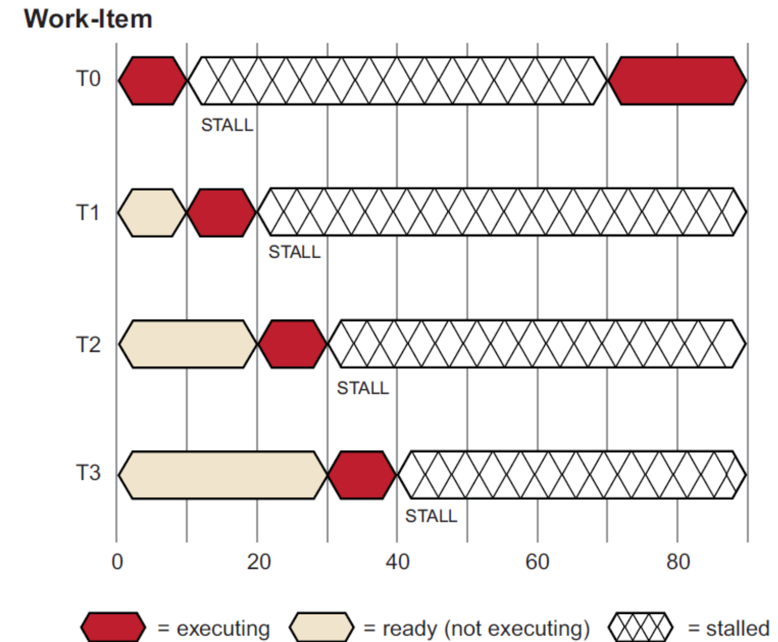
AMD GCN 1.0 Compute Unit sematikus ábra

Rejtett késleltetés

Előnyös



Előnytelen



NVIDIA Maxwell architektúra

MAXWELL
“GM204”
Top Level

- 5.2 Billion Transistors
- 2x performance vs GK104
- 16 SMM
- 2048 CUDA Cores
- 16 Geometry Units
- 128 Texture Units
- 64 ROP Units
- 256-bit GDDR5

NVIDIA Maxwell (GM204) sematikus ábra

NVIDIA Maxwell architektúra

MAXWELL SM

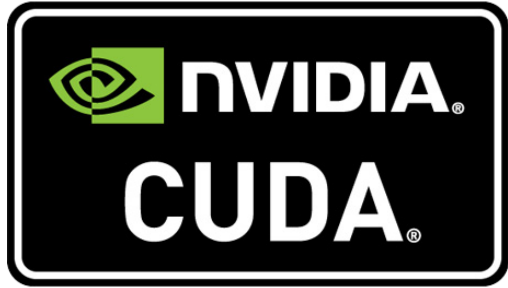
- 2x perf/watt vs GK104
- Improved scheduler
- New datapath organization
- +40% delivered perf
  per CUDA core

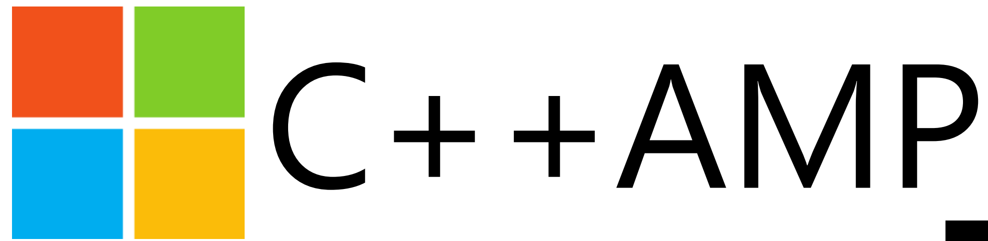
NVIDIA Maxwell SMM sematikus ábra

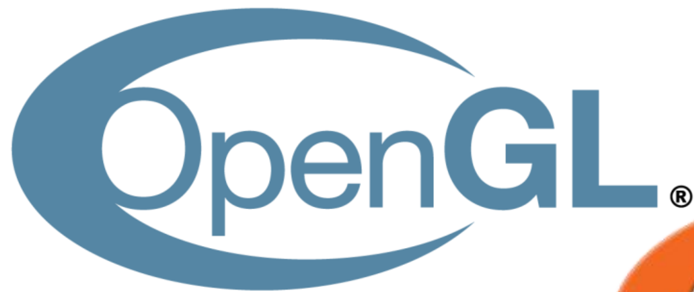
GPGPU API-k

A teljesség igénye nélkül

Egy állatkertre való API csokor







Compute Unified Device Architecture

- Az Nvidia Corporation válasza a fejlesztői presszióra, hogy kevesebb akadályt elgördítve lehessen grafikától független számolásokat végezni a GPU-n
- Első verziója 2007-ben jelent meg
- Két szintű hozzáférés
  - C meghajtó API
  - C++98 nyelvi kiterjesztés, amiben speciális operátorok és dekorátorok jelennek meg a gazda és eszköz oldali kód szétválasztására (CUDA 7.0-tól részleges C++11)
- Zárt technológia, a képességei a mindenkori Nvidia hardver képességeihez vannak igazítva

Open Computing Language

- Az Apple Incorporation kezdeményezése válaszul a CUDA sikerére, azonban a széleskörű érdeklődésre való tekintettel átengedték a fejlesztését a Khronos konzorciumnak
- Első verziója 2009-ben jelent meg
- Külön gazda és eszköz oldali nyelv
  - C API a gazda oldali kód számára
  - C-szerű nyelv az eszköz oldali kód számára (OpenCL 2.1-től statikus C++14)
- Nyílt technológia, bárki részt vehet a fejlesztésében és bárki implementálhatja

DirectX Compute Shader

- A Microsoft vezető 3-dimenziós grafikus API-jának GPGPU oldal ági kinövése
- 2009-ben jelent meg, A DirectX 11-ben debütált
- Triviális módon integrálódik a grafikus szerelőszalagba
  - A gazda oldali C++ API megörökli a grafikai felhasználás jellegzetességeit
  - Az eszköz oldali nyelve a HLSL (High-level Shading Language)
- DirectX 12?

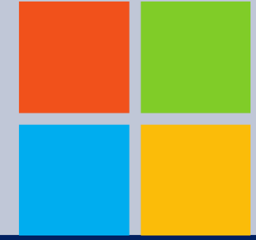
Wigner GPU Lab

Open Grahpics Library

Wigner
GPU Lab

- A Microsoft DirectX-ének nyílt változata, Khronos szabvány
- 2012-ben jelent meg, az OpenGL 4.3-ban debütált
- Triviális módon integrálódik a grafikus szerelőszalagba
  - A gazda oldali C++ API megörökli a grafikai felhasználás jellegzetességeit
  - Az eszköz oldali nyelve a GLSL (OpenGL Shading Language)

C++ Accelerated Massive Parallelism



- Tanulmány a masszív párhuzamosság szabványosítására, intrgrálására a C++ nyelvbe.
- 2012-es Microsoft fejlesztés, nyíltan implementálható
- C++ nyelvi kiterjesztés, amely a *concepts* nyelvi elem segítségével szeparálja a gazda és eszköz oldali kódot
- Nem határozza meg az implementáció bináris reprezentációját

Open Multi-Processing

OpenMP™

- Egy különálló konzorcium (OpenMP Architecture Review Board) fejleszti. Eredetileg CPU párhuzamosságot célzó technológia
- A 2013-as 4.0-ás verziójától kezdve támogatja a *gyorsítókra* történő feladat kiosztást
- Nyílt szabvány, nyíltan implementálható
- C, C++, Fortran nyelveket támogat preprocesszor direktívákon (`#pragma`) keresztül

Open Accelerators

OpenACC
Directives for Accelerators

Wigner GPU Lab

- Nvidia kezdeményezés egy OpenMP-szerű direktívák által vezérelt C, C++, fortran párhuzamos API létrehozására
- Nyílt szabvány, nyíltan implementálható
- 2013-ban egy tanulmány keretében megalapozták egy OpenMP-vel kompatibilis verzió alapjait, amely egy jövőbeni OpenMP szabványban teheti tiszteletét
- A legújabb 2.0-ás változatát idéntől támogathatja a GCC 5-ös verziója

SYCL

- Az OpenCL-t ért kritikák nyomán fejlesztett C++ sablon könyvtár
- Béta állapotban van, első verziója idén várható
- A gazda és eszköz oldali kód tisztán a sablon könyvtáron keresztül szeparálódik el
- A fordítás módját nem, de a köztes reprezentációt meghatározza
  - A motorháztető alatt az OpenCL által lefektetett technológiákra támaszkodik
  - A nevét (mint sarló) az OpenCL köztes nyelve (SPIR, mint lándzsa) után kapta

Nyílt szabványok FTW

Nyílt szabványok FTW

WIGNER GPU Lab

- A hordozhatóság nem le becsülendő
  - Teljesítmény tartalékok más platformokon
  - Időtállóság
  - Kompatibilis környezetek számossága
- A gyártók között felfedezett „közös nevező”
  - Absztrakciója a hardvernek
    - Tanuláshoz jó kiindulási pont
    - Könyvtárak fejlesztéséhez sem előnytelen
  - Az itt megszerzett tudás mindenhol használható lesz

OpenCL áttekintés

Avagy absztrakció Khronos-módra.

D. Berényi – M. F. Nagy-Egri

Mi szükség OpenCL-re?

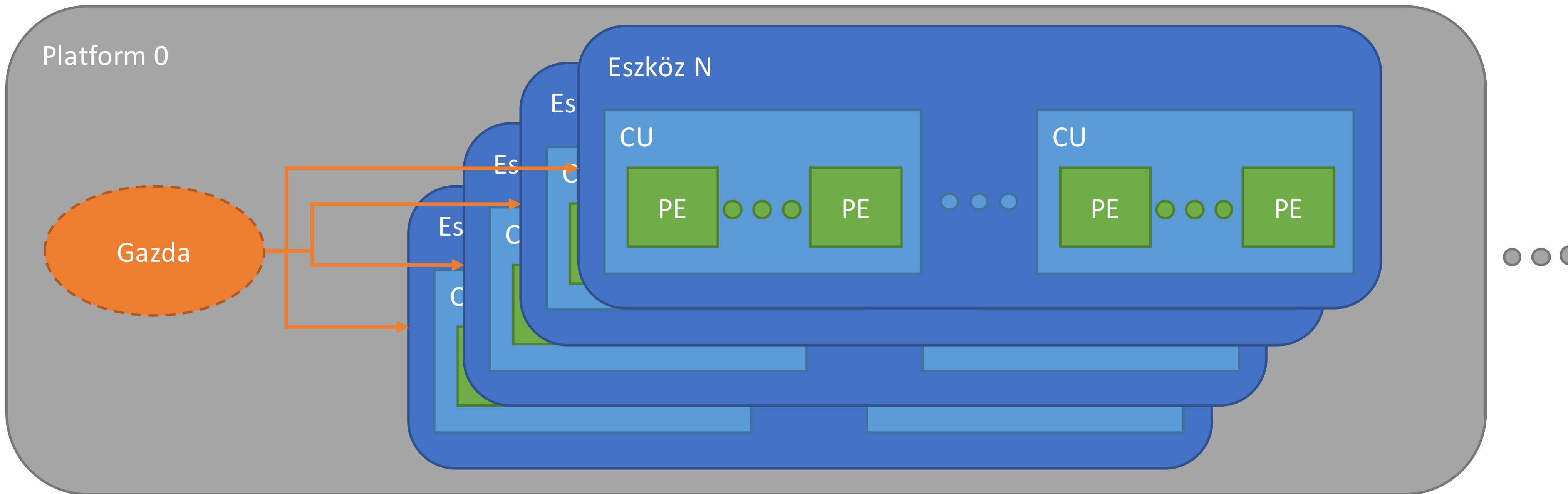
Wigner GPU Lab

- A hagyományos CPU-s eszközök (API-k, fordítók, stb.) szinte kivétel nélkül feltételeznek egy globálisan írható/olvasható memória címteret.
- A GPU felépítésénél hamar nyilvánvalóvá válik, hogy ennek az uniform címtérnek a szegmentálása révén drámaian egyszerűsödik a hardver
  - A felszabaduló erőforrások nagyobb teljesítményre fordíthatók
  - Jobb energiahatékonyság is elérhető
  - Egyes memória szeletek (címterek) csak bizonyos helyekről érhetők el. A hozzáférési szemantika egészen egzotikus is lehet.
- A modern hardverekben a GPU csak egy fajta célhardver
  - CPU, GPU, FPGA, DSP, crypto, stb.

Mit csinál OpenCL

- Az OpenCL absztrakcióiban NEM a hardveren van a hangsúly
- A párhuzamos struktúrák kerülnek előtérbe
  - Természetesen egy-az-egy megfeleltetés van a GPU részegységek és egyes absztrakciós elemek között
  - A tény, hogy FPGA-kra is nagyon jól fekszik mutatja azonban, hogy ennél többről van itt szó
- Alacsony szintű C API, amely ilyen formán képes más API-kkal is beszélgetni

OpenCL platform modell



- Egy platform, az egy implementációja az OpenCL szabványnak
- A futtató környezet áll egy gazda folyamatból és a platformok által enumerált eszközökből
  - Egy eszköz valahány számoló egységből áll (CU, Compute Unit)
  - Egy CU pedig valahány feldolgozó egységből áll (PE, Processing Element)

Mi egy OpenCL eszköz?

- Egy Compute Unit valahány végrehajtó egységből és egy írható/olvasható megosztott memóriából (Local Memory) áll. Egyedül képes futtatni egy teljes munkacsoportot (Work-group) és annak tagjait kiszolgálni a lokális memóriával.
- A munkacsoport olyan program szálak együttese, amelyek képesek hatékonyan szinkronizálni egymással, valamint adatokat megosztani egymással egy gyors, lokális memórián keresztül.
- Egy munkacsoport garantáltan egy végrehajtó egységre fog leképeződni, ugyanakkor semmi sem garantálja, hogy csak egy munkacsoport fog majd egy végrehajtón futni egyidejűleg.

OpenCL memória modell

Eszköz 1
CU 1
Eszköz L
CU 1
PE 1
PE N
privát
privát
lokális
CU M
PE 1
PE N
privát
privát
lokális
Globális/konstans gyorsító tár
Globális memória
Konstans memória
Gazda folyamat
Gazda oldali memória

- Az eszköz 4 külön memória névtérrel rendelkezik
  - Privát
  - Lokális
  - Globális/konstans
- A gazda folyamat hoz létre memória objektumokat, amik a gyorsító tárakba kerülnek

Wigner
GPU Lab

OpenCL C API

Ássunk mélyre

Platform választás, context létrehozás

```
cl_platform_id platform = NULL;
auto status = clGetPlatformIDs(1, &platform, NULL);

cl_device_id device = NULL;
status = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);

cl_context_properties cps[] = {CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
                               0};
auto context = clCreateContext(cps, 1, &device, 0, 0, &status);
auto queue = clCreateCommandQueueWithProperties(context, device, nullptr, &status);
```

- Lekérjük az első platformot amit találunk
- Lekérjük az első GPU-t, amit a platform támogat
- Létrehozunk egy kulcs-érték párokból álló tömböt a kontextusunkhoz
- Létrehozzuk magát a kontextust az egy darab eszközzel
- Létrehozunk egy parancslistát arra az egy eszközre

Forrás fájl betöltése programba

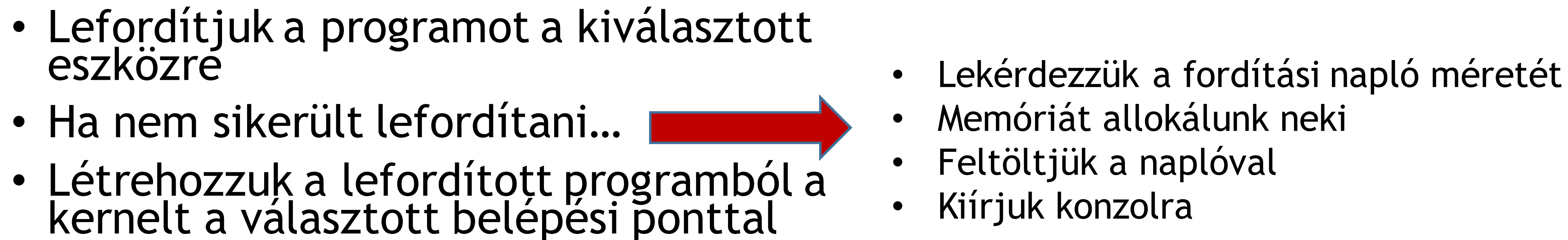
```
std::ifstream file(kernel_location);
std::string source(std::istreambuf_iterator<char>(file),
                   (std::istreambuf_iterator<char>()));
size_t      sourceSize = source.size();
const char* sourcePtr = source.c_str();
auto program = clCreateProgramWithSource(context,
                                         1,
                                         &sourcePtr,
                                         &sourceSize,
                                         &status);
```

- Létrehozunk egy fájl adatfolyamot
- Iterátorokkal beolvassuk a fájl teljes tartalmát (futás-időben)
- Létrehozunk belőle egy programot
  - A program tartalmazza majd a futtatható kerneleket
  - Egy program több kernelt is tartalmazhat

Program fordítás, kernel létrehozás

Wigner GPU Lab

```
status = clBuildProgram(program, 1, &device, "", nullptr, nullptr);
if (status != CL_SUCCESS) {
    size_t len = 0;
    status = clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0,
                                   nullptr, &len);
    std::unique_ptr<char[]> log = std::make_unique<char[]>(len);

    status = clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, len,
                                   log.get(), nullptr);
    printf("%s\n", log.get());
}
auto kernel = clCreateKernel(program, "squarer", &status);
```



- Lefordítjuk a programot a kiválasztott eszközre
- Ha nem sikerült lefordítani…
- Létrehozzuk a lefordított programból a kernelt a választott belépési ponttal

- Lekérdezzük a fordítási napló méretét
- Memóriát allokálunk neki
- Feltöltjük a naplóval
- Kiírjuk konzolra

Adatok feltöltése, kernel paraméterezés

WIGNER GPU Lab

```
std::array<float, 8> A{ 1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f, 7.0f, 8.0f };
std::array<float, 8> B{ 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
auto buffer_in = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                                A.size() * sizeof(float), A.data(), &status);
auto buffer_out = clCreateBuffer(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR,
                                 B.size() * sizeof(float), B.data(), &status);

status = clSetKernelArg(kernel, 0, sizeof(buffer_in), &buffer_in);
status = clSetKernelArg(kernel, 1, sizeof(buffer_out), &buffer_out);
```

- Létrehozunk két tömböt
  - Egy bemenetit és egy kimenetit (lehetne csak egyet is használni)
- Létrehozunk buffereket amelyek majd az eszközön tárolják az adatokat
  - A bufferek lusta módon fognak kikerülni az eszközre (első használatkor)
  - Nem feltétlen kell kiküldeni őket az eszközre (hozzáférési zászlókból következhet)

Adatok feltöltése, kernel paraméterezés

```
std::array<float, 8> A{ 1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f, 7.0f, 8.0f };
std::array<float, 8> B{ 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
auto buffer_in = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                                A.size() * sizeof(float), A.data(), &status);
auto buffer_out = clCreateBuffer(context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR,
                                 B.size() * sizeof(float), B.data(), &status);

status = clSetKernelArg(kernel, 0, sizeof(buffer_in), &buffer_in);
status = clSetKernelArg(kernel, 1, sizeof(buffer_out), &buffer_out);
```

- Hozzárendeljük a létrehozott puffereket a kernel függvény argumentumaihoz
  - Nincs típus ellenőrzés, memcpy történik a kettő között
  - A sizeof operátor az argumentum méretére utal, nem a buffer tartalmának a méretére
  - Egyedül a kernel és az indexedik argumentum létezése ellenőrződik

Végrehajtás

```
size_t thread_count = A.size();
status = clEnqueueNDRangeKernel(queue, kernel, 1, nullptr, &thread_count, nullptr,
                                0, nullptr, nullptr);
status = clEnqueueReadBuffer(queue, buffer_out, false, 0, B.size() * sizeof(float),
                             B.data(), 0, nullptr, nullptr);
status = clFinish(queue);

std::for_each(B.begin(), B.end(), [](float x) { std::cout << x << "\n"; });
```

- Elindítunk annyi work-itemet, ahány elemű a tömbünk
- Ezután visszakérjük az eredményt gazda oldali memóriába
- Megvárjuk, míg ezek a műveletek elkészülnek
  - Alapértelmezetten a parancslistába rakott műveletek sorrend őrzők
  - A listában szereplő műveletek azonban késleltetve hajtódnak végre (futószalag-szerű végrehajtás)

Mit tud és mit nem ez a program?

- Meg lehet rajta keresztül érteni az API elemek egymásra épülését

- Hibakezelést szinte teljesen nélkülözi
  - Mi van, ha az első platform nem létezik?
  - Mi van ha nincs rajta GPU?
  - Mi van, ha a kernel forrás fájlt nem tudtuk betölteni?
  - Mi van, ha bufferek nem férnek el memóriában?
  - Mi van, ha rossz számú kernel példányt indítottunk el?
- Nem mellesleg minden API erőforrást elszivárogtat

OpenCL C++ API

Citius, altius, fortius

D. Berényi – M. F. Nagy-Egri

Egy szebb és jobb világ

WIGNER GPU Lab

- Az OpenCL C++ nélkül olyan, mint a Dallas Bobby nélkül…
  - Lehet hassználni, de minek?
  - Lásd itt
- Csomagoljuk be idiomatikus C++ba
  - RAII-val garantáljuk az API erőforrások helyes felszabadítását
  - Típus ellenőrizzük le, hogy helyesen paraméterezzük-e fel a kernelt
  - A tömb műveletek ne fájjanak („mennyi-foglal-feltölt” minta)
  - Hibakódokat ne kelljen minden sarkon ellenőrizni (kód felfújás)
  - Ne legyen nagyon idegen (a lehetőségekhez mérten)

Madártávlati felépítés


```
int main()
{
    try // Any error results in program termination
    {
        ...
    }
    catch (cl::BuildError error) // If kernel failed to build
    {...}
    catch (cl::Error error) // If any OpenCL error happes
    {...}
    catch (std::exception error) // If STL/CRT error occurs
    {...}

    return EXIT_SUCCESS;
}
```

A programunk legnagyobb része itt lesz. Nézzük is meg!

C++ kivételek: beépített hibakezelési módszer

Platform választás

```
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);

for (const auto& platform : platforms)
    std::cout << "Found platform: " << platform.getInfo<CL_PLATFORM_VENDOR>() <<
std::endl;
```

- Dinamikus tömb létrehozása
  - Malloc kivételt dobhat
- OpenCL Installable Client Driver lekérdezés: milyen platformok léteznek?
  - API hívás ha hiba kóddal tér vissza, a wrapper kivételt dob
- Konzolra kiírjuk mindnek a nevét a felhasználó megelégedésére.
  - Ha a platform rosszul működik (vagy rossz a varázs define): kivétel

Platform választás

```
// Choose platform with most DP capable devices
auto plat = std::max_element(platforms.cbegin(), platforms.cend(),
                             [](const cl::Platform& lhs, const cl::Platform& rhs)
{
    auto dp_counter = [](const cl::Platform& platform)
    {
        std::vector<cl::Device> devices;
        platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);

        return std::count_if(devices.cbegin(), devices.cend(),
                             is_device_dp_capable);
    };

    return dp_counter(lhs) < dp_counter(rhs);
});
```

Szedjük darabjaira ezt a kód részletet!

Platform választás

```
// Choose platform with most DP capable devices
auto plat = std::max_element(platforms.cbegin(), platforms.cend(),
                             [](const cl::Platform& lhs, const cl::Platform& rhs)
{
    auto dp_counter = [](const cl::Platform& platform)
    {
        std::vector<cl::Device> devices;
        platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);

        return std::count_if(devices.cbegin(), devices.cend(),
                             is_device_dp_capable);
    };

    return dp_counter(lhs) < dp_counter(rhs);
});
```

Keressük azt a platformot, ami a legtöbb vmilyen eszközt ismeri.

Platform választás

```
// Choose platform with most DP capable devices
auto plat = std::max_element(platforms.cbegin(), platforms.cend(),
                             [](const cl::Platform& lhs, const cl::Platform& rhs)
{
    auto dp_counter = [](const cl::Platform& platform)
    {
        std::vector<cl::Device> devices;
        platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);

        return std::count_if(devices.cbegin(), devices.cend(),
                             is_device_dp_capable);
    };

    return dp_counter(lhs) < dp_counter(rhs);
});
```

A rendezési feltétel egy függvény eredménye alapján rendez.

Platform választás

```
// Choose platform with most DP capable devices
auto plat = std::max_element(platforms.cbegin(), platforms.cend(),
                             [](const cl::Platform& lhs, const cl::Platform& rhs)
{
    auto dp_counter = [](const cl::Platform& platform)
    {
        std::vector<cl::Device> devices;
        platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);

        return std::count_if(devices.cbegin(), devices.cend(),
                             is_device_dp_capable);
    };

    return dp_counter(lhs) < dp_counter(rhs);
});
```

Najó, de ez hogyan néz ki?

Egy platformon leszámolja a beszédes feltételnek eleget tevőket.

Platform választás

WIGNER GPU Lab

```
// Checks weather device is DP calable or not
bool is_device_dp_capable(const cl::Device& device)
{
return
    device.getInfo<CL_DEVICE_EXTENSIONS>().find("cl_khr_fp64") != std::string::npos;
}
```

- Hát ennél sokkal egyszerűbb nemigen lehet.
- Keressük az összes támogatott kiterjesztések stringjében azt az egy szabványos kiterjesztést, amire kíváncsiak vagyunk.
- Ha a stringben a helye nem egyenlő a „nem létező pozícióval” (no pos), akkor megtaláltuk valahol, tehát támogatott.

Platform választás

```
// Choose platform with most DP capable devices
auto plat = std::max_element(platforms.cbegin(), platforms.cend(),
                             [](const cl::Platform& lhs, const cl::Platform& rhs)
{
    auto dp_counter = [](const cl::Platform& platform)
    {
        std::vector<cl::Device> devices;
        platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);

        return std::count_if(devices.cbegin(), devices.cend(),
                             is_device_dp_capable);
    };

    return dp_counter(lhs) < dp_counter(rhs);
});
```

Most már reméljük tiszta, mi történik itt.

Platform választás


```
if (plat != platforms.cend())
    std::cout << "Selected platform: " << plat->getInfo<CL_PLATFORM_VENDOR>()
        << std::endl;
else
    throw std::runtime_error{ "No double-precision capable device found." };
```

- Biztos ami biztos, a felhasználót értesítsük, hogy melyik platformot választottuk ki végül.
- Ha nincs egy ilyen eszköz sem, akkor egy kivétellel eltűnünk a süllyesztőben.

Eszköz választás

```
// Obtain DP capable devices
std::vector<cl::Device> devices;
plat->getDevices(CL_DEVICE_TYPE_ALL, &devices);

std::remove_if(devices.begin(), devices.end(),
               [](const cl::Device& dev) { return !is_device_dp_capable(dev); });

cl::Device device = devices.at(0);

std::cout << "Selected device: " << device.getInfo<CL_DEVICE_NAME>() << std::endl;
```

- Az adott platform összes eszközét lekérjük…
- és most nem számolunk, hanem szűrünk.
- Eztán kivesszük a nulladik elemet
  - Ha kozmikus véletlen folytán nem volna nulladik elem: kivétel

Context és parancslista létrehozás


```
// Create context and queue
std::vector<cl_context_properties> props{ CL_CONTEXT_PLATFORM,
    reinterpret_cast<cl_context_properties>((*plat)()),
    0 };

cl::Context context{ devices, props.data() };

cl::CommandQueue queue{ context, device, cl::QueueProperties::Profiling };
```

- Context létrehozáshoz tulajdonság tömb kell (kulcs-érték pár)
  - Minimálisan azt az egyet mondjuk meg, melyik platformhoz tartozik
  - cl_platform_id (pointer típus) castolunk egy másik pointer típusra
    - Más API-kkal lehet itt beszélgetni (OpenGL interop, lásd később)
  - Kb. az egyetlen valid használata a reinterpret_castnak
- Parancslista létrehozásánál nincs ilyen voodoo…

Kernel kód betöltés és fordítás

```
// Load program source
std::ifstream source_file{ kernel_location };
if (!source_file.is_open())
   throw std::runtime_error{ std::string{"Cannot open source: "} + location };

// Create program and kernel
cl::Program prog{ context,
                  std::string{ std::istreambuf_iterator<char>{ source_file },
                               std::istreambuf_iterator<char>{} } };
prog.build({ device }, "-cl-std=CL1.0"); // Simplest kernel syntax

auto vecAdd = cl::KernelFunctor<cl_double, cl::Buffer, cl::Buffer>(prog, "vecAdd");
```

- Futásidőben betöltjük a forrásfájlt és lefordítjuk a kiválasztott eszközre.
- Típusbiztos függvény objektumot hozunk létre (funktor).

Gazda oldali memória allokáció

Wigner GPU Lab

```
// Init computation
const std::size_t chainlength = std::size_t(std::pow(2u, 20u));
// 1M, cast denotes floating-to-integral conversion,
//     promises no data is lost, silences compiler warning

std::valarray<cl_double> vec_x( chainlength ),
                         vec_y( chainlength );
cl_double a = 2.0;
```

- Foglaljunk le két 1 millió elemű tömböt (vélhetően memória szemét kezdeti értékekkel)
- A típus sejteti, hogy ezekkel valami huncutságot fogunk csinláni
  - Saxpy nagyvektor műveletet $y = a \cdot x + b$
- Jegyezzük meg, hogy API-specifikus típust használunk
  - Ha egy adott API-val dolgozunk, tanácsos annak a típusait használni

Gazda oldali memória inicializálása

```
// Fill arrays with random values between 0 and 100
auto prng = [engine = std::default_random_engine{},
             dist = std::uniform_real_distribution<cl_double>{-100.0, 100.0 }]()
mutable { return dist(engine); };

std::generate_n(std::begin(vec_x), chainlength, prng);
std::generate_n(std::begin(vec_y), chainlength, prng);
```

- Az STL véletlen generátorait használjuk:
  - Kell egy eloszlás, amiből véletlen mintát szeretnénk
  - Kell egy véletlen bit sorozatot előállító forrás (engine)
  - Az egyiket odaadjuk a másiknak és kész!
- Ha egy lambda módosíthatja a saját állapotát, azt jelezni kell
- Véletlen számokkal tömböt feltölteni? Algoritmusok!

Eszköz oldali memória allokáció/init

```
cl::Buffer buf_x{ context, std::begin(vec_x), std::end(vec_x), true },
           buf_y{ context, std::begin(vec_x), std::end(vec_x), false };

// Explicit (blocking) dispatch of data before launch
cl::copy(queue, std::cbegin(vec_x), std::cend(vec_x), buf_x);
cl::copy(queue, std::cbegin(vec_x), std::cend(vec_x), buf_y);
```

- OpenCL bufferek létrehozása az adatmozgatás leírásához
  - A buffer NEM egy adott eszközön lefoglalt folytonos memóriát jelent
  - Eszközről eszközre (és gazda oldalra költözhet)
  - Az API leköveti hol írták utoljára, kinek van meg a legutóbbi verziója
  - „Lusta” adatmozgatás, de lehet expliciten is (ahogy azt tesszük itt)
- Context és forrás adat meghatározás, mozgatás

Eszköz oldali számolás

```
// Launch kernels
cl::Event kernel_event{ vecAdd(cl::EnqueueArgs{ queue, cl::NDRange{ chainlength } },
                        a, buf_x, buf_y) };

kernel_event.wait();

std::cout << "Device (kernel) execution took: „ <<
    util::get_duration<CL_PROFILING_COMMAND_START,
                       CL_PROFILING_COMMAND_END,
                       std::chrono::microseconds>(kernel_event).count() <<
    " us." << std::endl;
```

- Berakunk chainlength darab számú vecAdd invokációt az adott parancslistába adott paraméterekkel
  - Típus ellenőrzés történik, hogy helyesen paraméterezünk-e
  - Megvárjuk, míg a számolás befejeződik

Időméréshez kis segítség

WIGNER GPU Lab

```
template <cl_int From, cl_int To, typename Dur = std::chrono::nanoseconds>
auto get_duration(cl::Event& ev)
{
    return std::chrono::duration_cast<Dur>(std::chrono::nanoseconds{
        ev.getProfilingInfo<To>() - ev.getProfilingInfo<From>() });
}
```

- Egy bejövő OpenCL esemény két adott állapota között eltelt időt adja vissza STL kompatibilis formában
  - Visszatérési érték függ a bemenettől, inkább elhagyjuk
  - Az OpenCL API alapból nanoszekundumokban mér, így ezt választjuk alapértelmezett idő mértéknek.

Referencia számolás

```
// Compute validation set on host
auto start = std::chrono::high_resolution_clock::now();

std::valarray<cl_double> ref = a * vec_x + vec_y;

auto finish = std::chrono::high_resolution_clock::now();

std::cout << "Host (validation) execution took: " <<
    std::chrono::duration_cast<std::chrono::microseconds>(finish - start).count() <<
    " us." << std::endl;
```

- Referencia számolás egy sor
- Időmérés STL időzítőkkel történik

Validáció

```
// (Blocking) fetch of results
cl::copy(queue, buf_y, std::begin(vec_y), std::end(vec_y));

// Validate (compute saxpy on host and match results)
auto markers = std::mismatch(std::cbegin(vec_y), std::cend(vec_y),
                             std::cbegin(ref), std::cend(ref));

if (markers.first != std::cend(vec_y) || markers.second != std::cend(ref))
    throw std::runtime_error{ "Validation failed." };
```

- Visszakérjük az eredményt az eszközről gazda oldali memóriába
- Megkeressük páronként az első olyan helyet, ahol az eredmény tömb eltér a referencia tömbtől
- Ha ezek a helyek külön-külön nem esnek egybe a tömbök utolsó utáni elemével (STL konvenció a „nem talált”-ra): kivétel.

Algoritmus könyvtárak

Magasabb rendű függvények, mint építőkockák

Alacsonyan fekvő gyümölcsök

- A legtöbb GPGPU API-ba vagy be vannak építve a leggyakrabban használt algoritmus primitívek, vagy készül hozzájuk könyvtár
- Ezek mind valamelyik API-ra támaszkodnak a végrehajtás szintjén
  - Megöröklik az alatta lévő API összes korlátozását (OS, HW, stb.)
- A nagyon egyszerű feladatok pillanatok alatt megoldhatók
- A bonyolultabbak sajnos szinte lehetetlenek (nem törvényszerű!)

Thrust

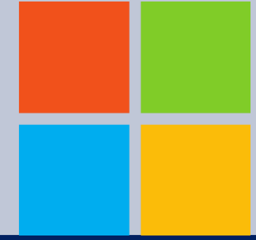
WIGNER GPU Lab

- CUDA sablon algoritmus könyvtár
- Kényelmesen interfészelhető az STL-lel magával
- Mivel a CUDA-ra támaszkodik, ezért
  - Csak Nvidia hardverrel használható
  - Csak nvcc fordítóval fordítható
- Legstabilabb, flexibilis környezet

Thrust

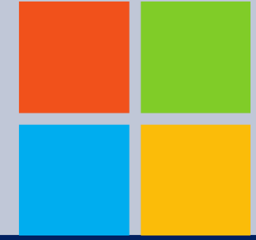
```
const float a = 2.0f;
thrust::device_array<float> x(size);
thrust::device_array<float> y(size);

thrust::sequence(x.begin(), x.end());
thrust::sequence(y.begin(), y.end());

thrust::transform(x.begin(), x.end(), y.begin(), y.begin(),
__host__ __device__ [=]( const float& x, const float& y )
{
    return a * x + y;
} );
```

C++AMP Algorithms Library



- C++AMP sablon algoritmus könyvtár
- Kényelmesen interfészelhető az STL-lel magával
- Mivel az AMP-ra támaszkodik, ezért
  - Csak AMP-kompatibilis fordítóval fordítható
  - Cserébe bárhol, bármin futtatható
- Béta verzió, cserébe open-source

C++AMP Algorithms Library



```
const float a = 2.0f;
concurrency::array<float> x(size);
concurrency::array<float> y(size);
array_view<float> x_av(x);
array_view<float> y_av(y);

amp_stl_algorithms::iota(begin(x_av), end(x_av), 1.0f);
amp_stl_algorithms::iota(begin(y_av), end(y_av), 1.0f);

amp_stl_algorithms::transform(begin(x_av), end(x_av), begin(y_av), begin(y_av),
[=]( const float& x, const float& y ) restrict(amp,cpu)
{
    return a * x + y;
} );
```

Bolt Template Library

BOLT

Wigner GPU Lab

- C++ sablon algoritmus könyvtár
- Kényelmesen interfészelhető az STL-lel magával
- Több implementációra támaszkodik egyszerre
  - Intel Thread Building Blocks közönséges CPU-ra
  - OpenCL CPU/GPU-ra egyaránt
  - Az interfésze magán hordozza az implementációk közös nevezőjét
- Open-source

Bolt Template Library

```
BOLT_FUNCTOR(saxpy_functor,
struct saxpy_functor
{
      const float a;

      saxpy_functor(float _a) : a(_a) {}

      float operator()(const float& x, const float& y) const
      {
            return a * x + y;
      }
};
);
```

Bolt Template Library

BOLT

Wigner GPU Lab

```
const float a = 2.0f;
bolt::cl::device_vector<float> x(size);
bolt::cl::device_vector<float> y(size);

std::iota(x.begin(), x.end(), 1.0f);
std::iota(y.begin(), y.end(), 1.0f);

bolt::cl::transform(x.begin(), x.end(), y.begin(), y.begin(), saxpy_functor(a));
```