

OpenCL alapú eszközök verifikációja és validációja a gyakorlatban

Fekete Tamás

2015. December 3.

Szoftver verifikáció és validáció tantárgy



Department of
Automation and
Applied Informatics

Áttekintés

- Miért és mennyire fontos a megfelelő validáció és verifikáció OpenCL alapú eszközök esetében a tapasztalataim alapján?
 - > Miért nagy kihívás?
 - > Milyen gyakorlati megoldásokat találtam
- **A folyamatos Verifikáció és validáció ma már a fejlesztés és kutatás során is alapvető eszköz**, hiszen az igények folyamatosan változnak menet közben (*pl. kutatás eredmények szerint vagy Agilis szoftver fejlesztés során ügyfél kérésére*)
- Túlságosan formális módszerek algoritmusok fejlesztésére/kutatásra
 - > Lassú és összetettebb problémák esetén túl bonyolult
 - > Másnak nehéz megérteni, javítani
- **Gyakorlatorientált verifikációt és validációt vizsgálom ebben a munkában**

OpenCL keretrendszer

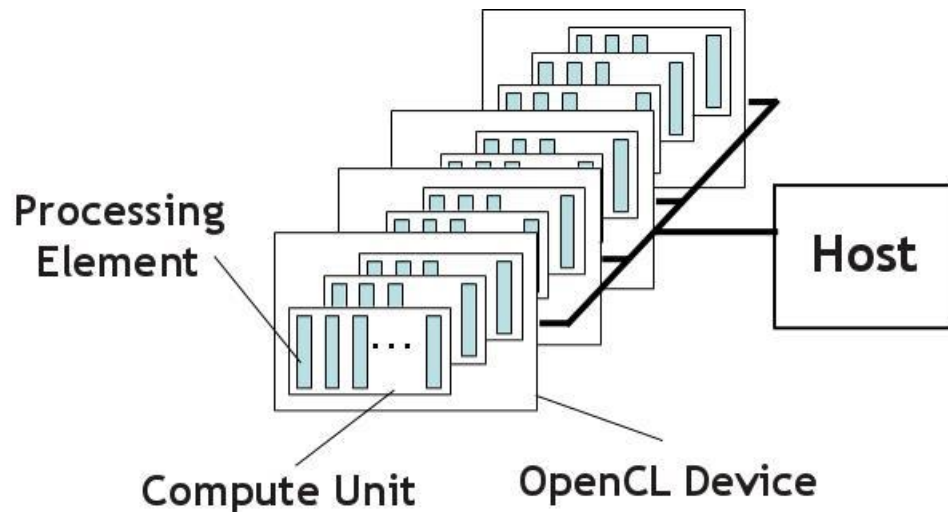
- Csak egy interfészt definiál (*C nyelven*), Khronos Group adja ki
- Célja több hardver egy közös, egységes interfészen történő elérése
- A hardvergyártók maguk végzik a megvalósítást



- Eltérés lehet a működésben
 - > Elsősorban a **teljesítmény** optimalizációjában mutatkozik meg
 - > A gyártók **különböző verzióját** támogatják a keretrendszernek (pl. NVIDIA GPU: 1.2, AMD GPU: 2.1)
 - > Többnyire *grafikus vezérlők esetén használják, de támogatják FPGA és DSP gyártók is*

OpenCL programozási modell (2/1)

- A verifikáció és validáció szempontjából is lényeges (ábra)
- Elsősorban **párhuzamosan futó algoritmusok** fejlesztésére készült
- Fontos, hogy az *OpenCL kód és adat* nem a „host” memóriában van/fut, hanem az OpenCL keretrendszer által kezelt célhardveren
 - > Végrehajtás előtt át kell másolni az adatokat (opcionális) és a binárist

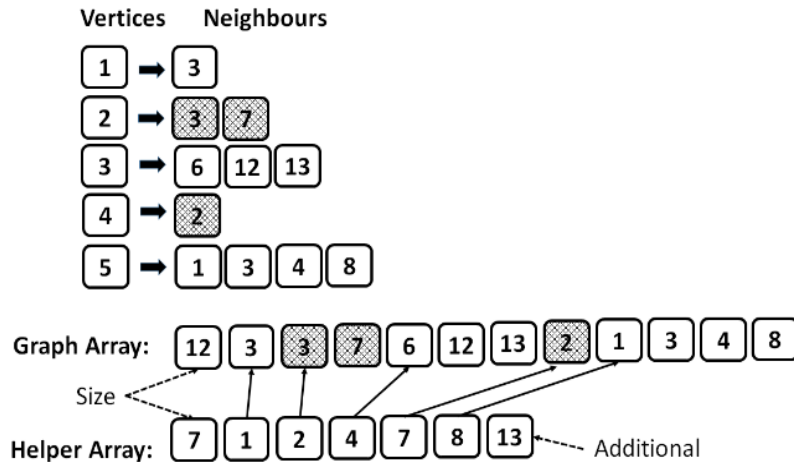


OpenCL programozási modell (2/2)

- A célhardveren futó forráskód nem része a „host” forráskódjának, általában külön fájlban is van
- Futásidőben a „host” az OpenCL keretrendszer interfészein keresztül az OpenCL forrásból készít egy binárist és tölti le a céleszközre
- Nincs lehetőség a „szokásos” debuggolási lehetőségekre
 - > pl. tetszőleges IDE-k általi „lépekedés” a kódban
 - > A legtöbb gyártó biztosít valamilyen debuggolási lehetőséget, de gyakran komplikáltak és egymástól eltérőek
- Az OpenCL-ben írt kódot (*függvényét kernelnek hívjuk*), C nyelven kell írni kevés megszorítás mellett

Be- és kimeneti adatok (OpenCL 1.2)

- Csak 1 dimenziós tömböt lehet másolni (*mutató tömb sem megengedett*)
- A bemeneti adatokat le kell képezni
 - > Verifikálni és validálni kell az alkalmazott módszert
 - > Funkcionalitást tesztelem, visszafele is átalakítom az adatokat
- Kimeneti adatokat szintén vissza kell másolni a hostra, ahhoz, hogy fel lehessen dolgozni azokat
 - > Tesztadatokat kell visszamásolni és feldolgozni
- ADATÁRAMLÁS ellenőrzése! Ehhez szoftver adatmodelt készíték.



Tervezés (2/1)

- Gyakorlati példa: mintakeresés gráfokban
- Automata és egyéb matematikai módszerek
 - > Automata implementáció készült is
- **Interface alapú tervezés**, nemcsak a legfelső, hanem közbenső szinteken is interfacekkel dolgozom a host kódban
 - > Jól definiálható elvárások az egyes modulokkal szemben
 - > Belső implementáció változtatása esetén a ráépülő modulokat nem kell javítani
- Pszeudókód készítése, mivel itt a kernel algoritmus a legfontosabb és a legkritikusabb elem

Tervezés (2/2)

- Memória management fontos a kernel kódban (sok hibalehetőség):
 - > dinamikusan nem lehet memóriát allokálni
- Többnyire a programozási modellekben 4 memóriaterületet különböztetünk meg:
 - > host, globális, privát és konstans értékeknek *(az utolsó az eszköz globális memóriaterületén belül)*
- Skálázhatóság lényeges szempont, mivel nem lehet tudni, hogy mikor fogy el a memória
 - > Verifikálok és tesztelem **(kézzel korlátozom a foglalható memória méretét) -> ugyanolyan eredménynek kell lennie korlátozás nélkül is**
 - > A grafikus vezérlők többnyire kisebb memóriával rendelkeznek mint a host

Hibakezelés

- Elsődlegesen az operációs rendszer egy idő után, ha nem válaszol a grafikus vezérlő, akkor „reset”-eli azt és tájékoztat a „hibáról”
- Az OpenCL keretrendszer is tud hibás működést észrevenni és hibakóddal visszatérni
 - > A specifikáció tartalmazza a hibakódok jelentéseit
 - > Gyakran nem elég specifikusak
- **Működés közben saját hibakódokat definiálok és kimenő paraméterként visszaadom**
 - > A kódnak le kell futnia
 - > Funkcionális hibák megtalálására alkalmas
 - > *pl. túl kicsi a kimeneti tömb, nincs több hely benne*

Statikus és dinamikus ellenőrzés

- Statikus kódelemzésre ingyenes online ellenőrzés is elérhető, de számtalan ingyenes alkalmazás is van
 - > A jobbak már fizetősek
- Két dinamikus szintű tesztelést végzek:
- Unit/modul tesztek: funkcionális követelmények ellenőrzésére használom, valamint fejlesztés során
- Esettanulmányok: a nem funkcionális követelmények ellenőrzésére
 - > Nagy bemeneti adathalmaz
 - > Futási idő és skálázhatóság vizsgálata

Statikus kód ellenőrzés

- Számptalan program, pl. *CLEditor*
- Online ellenőrzés, pl. *Stream Computing*

```
697  kernel void
698  MonoFractalArray2dU(
699  __global uchar4 *output,
700  const float2 bias,
701  const float2 scale,
702  const float lacunarity,
703  const float increment,
704  const float octaves,
705  const float amplitude)
706  {
707  {
708  int2 coord = (int2)(get_global_id(0), get_global_id(1));
709
710  int2 size = (int2)(get_global_size(0), get_global_size(1));
711
712  float2 position = (float2)(coord.x / (float)size.x,
713  coord.y / (float)size.y);
714
715  float2 sample = (position + bias);
716
717  float value = monofractal2dU(sample, scale.x, lacunarity, increment, octaves);
718
719  float4 result = (float4)(value, value, value, 1.0f) * amplitude;
720  }
```

Description	Line	Column
type qualifier specified more than once	7	12
type qualifier specified more than once	8	12
variable "remainder" was declared but never referenced	509	11
variable "sample" was declared but never referenced	510	11
variable "remainder" was declared but never referenced	582	11

Stay up-to-date: [Twitter](#) [LinkedIn](#) [RSS](#) - Get in touch today: [Phone](#)

STREAM COMPUTING

Performance Engineers

[Company](#) [Software Development](#) [Consultancy](#) [Training](#)

After the LEAP-conference I'll extend this article - till then I'm too time-limited. For now I wanted to share the online version with you, especially with the people who will attend the [tutorial](#) at LEAP. Be sure to check out the [GPUVerify website](#) and [paper](#) to learn more about this fantastic tool!

Using version 2015-09-23.

Number of groups: x

Local size: x

Mode: verify (default)
 bug-finding

Language: OpenCL (default)
 CUDA

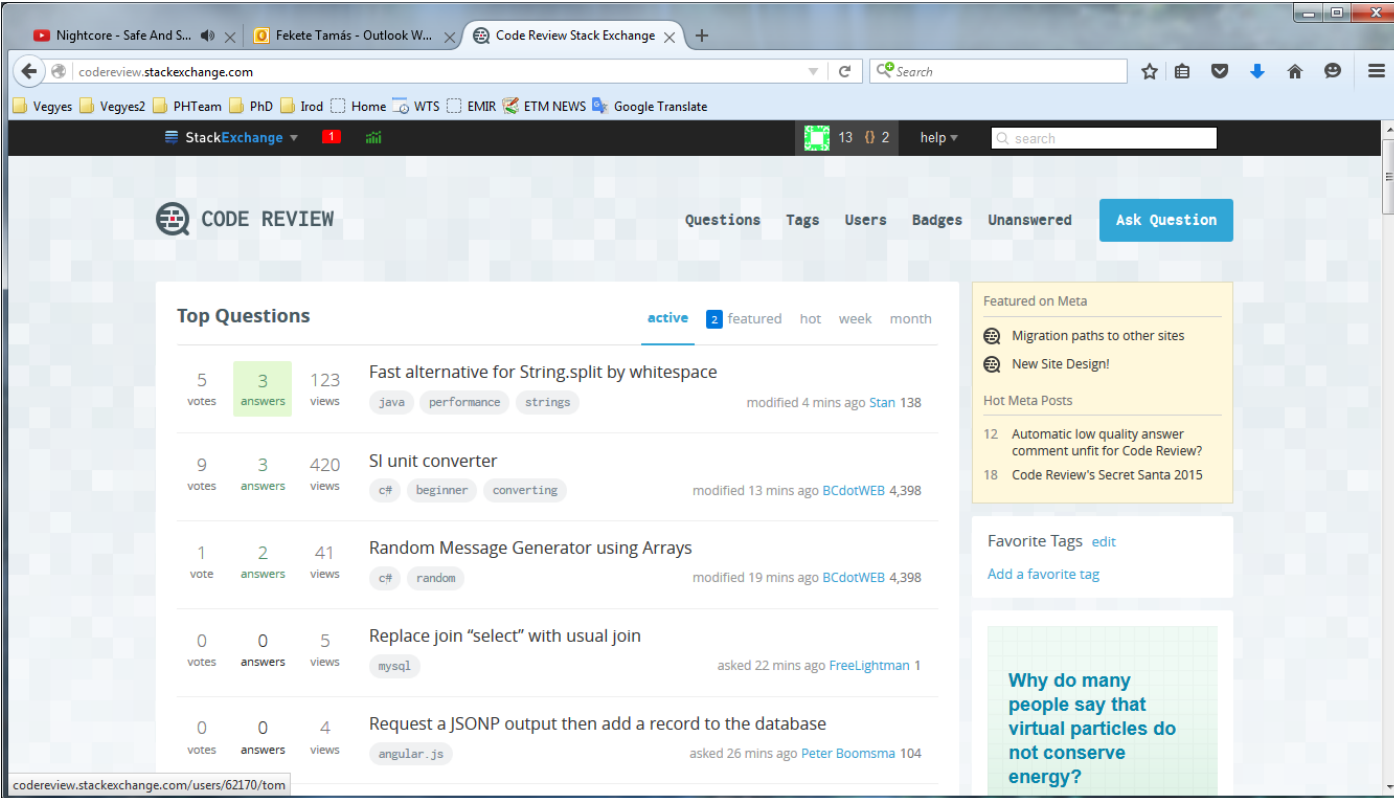
OpenCL kernel:

```
__kernel void add_neighbour(__local int *A, int offset) {
    int tid = get_local_id(0);
    A[tid] += A[tid + offset];
}
```

Kód ellenőrzés - review

- Kedvencem:

> <https://codereview.stackexchange.com>



The screenshot shows the Code Review Stack Exchange website. The page features a navigation bar with links for Questions, Tags, Users, Badges, and Unanswered, along with an Ask Question button. The main content area displays a list of top questions, each with its vote count, answer count, view count, tags, and modification time. The questions listed are:

Votes	Answers	Views	Question Title	Tags	Modified
5	3	123	Fast alternative for String.split by whitespace	java, performance, strings	modified 4 mins ago Stan 138
9	3	420	SI unit converter	c#, beginner, converting	modified 13 mins ago BCdotWEB 4,398
1	2	41	Random Message Generator using Arrays	c#, random	modified 19 mins ago BCdotWEB 4,398
0	0	5	Replace join "select" with usual join	mysql	asked 22 mins ago FreeLightman 1
0	0	4	Request a JSONP output then add a record to the database	angular.js	asked 26 mins ago Peter Boomsma 104

On the right side of the page, there is a sidebar with sections for 'Featured on Meta' (Migration paths to other sites, New Site Design!), 'Hot Meta Posts' (Automatic low quality answer comment unfit for Code Review?, Code Review's Secret Santa 2015), 'Favorite Tags' (Add a favorite tag), and a featured article titled 'Why do many people say that virtual particles do not conserve energy?'.

Unit és modul tesztek

- Funkcionális követelmények ellenőrzésére használok, példák (határérték ellenőrzése):
 - > „Smoke” teszt új ötletek/implementáció elején
 - > Egy elemet csak egyszer találjon meg keresés esetén
 - > Ha nincs eredmény, akkor is jól működjön
 - > Az összes extrém esetre egy-egy egyszerű példa
 - > Negatív tesztesetek a tesztkörnyezet tesztelésére
- Rövid kód, gyorsan lefut, sok könyvtárban van támogatás hozzá (*pl. Boost*)

Esettanulmányok

- A nem-funkcionális követelmények tesztelésére használom
- Nincsenek tesztesetek
 - > Időbeli és memóriabeli korlátozás
 - > Host verzióval hasonlítom össze
- Nagy bemeneti adathalmaz:
 - > Idő és memóriahasználat mérése
 - > Mérési pontok elhelyezése
- **A legfontosabb szoftver jellemzők vizsgálata itt történik:**
 - > skálázhatóság, megbízhatóság, hibakezelés, platform függetlenség

Teszt alapú fejlesztés (TDD)

- A unit és a modul tesztek már előre, az implementálás előtt elkészülnek
- Egyszerre kezelem a teszteket és a célkódot, így a folyamatos futtatással folyamatosan kapok visszajelzést, hogy a kód jól működik-e
- *Egyszerű példákat gyorsabban lehet debuggolni*
- Kihívás: előre pontosan tudni kell, hogy mit várok el az algoritmustól
 - > Kisebb algoritmusokkal szemben könnyen lehet előre tudni az elvárásokat

Automatizálás

- A Microsoft is támogat regressziós tesztek:
 - > Ingyenesen lehet futtatni a TFS alatt tárolt forráskódokra
- „Egyszemélyes” fejlesztésnél nem hatékony, az eredmények kiértékelése alatt a teszt kézzel is könnyen futtatható
- Kisebb alkalmazások, algoritmusok esetén szintén nem szükséges

Dokumentáció

- **Előre készülő tervek**

- > Optimalizáció: a túlságosan rövid és a túlságosan hosszú tervezés is jelentősen megnöveli a megvalósítást

- **Utólagos használatra**

- > Segítik a régen írt kódokat megérteni
 - > Hibajavítás során már a dokumentáció erősen tudja sejtetni, hogy hol található a hiba

Köszönöm a figyelmet!