Lastenheft - Weiterentwicklung des “Gorynych” Projekts

Struktur

- Projektbeschreibung
- Ist-Zustand
- Soll-Zustand
- Systemarchitektur
- Funktionale Anforderungen
- Nicht-Funktionale Anforderungen
- Benutzbarkeit
- Wartbarkeit
- Zuverlässigkeit
- Abnahmekriterien

Überblick über das Projekt:

“gorynych” ist eine templateorientierte Abstraktionsschicht zur Durchführung von beschleunigten Berechnungen auf heterogenen Systemen. Mit diesem Framework ist es möglich, ein einziges Mal den Algorithmus zu implementieren und von der automatischen Abbildung auf verschiedene Prozessor-Fähigkeiten zu profitieren.

Ohne “gorynych” müsste man für jeden Prozessor-Fähigkeit den Algorithmus neu schreiben, dies hat erhöhte Entwicklungs- und Wartungskosten zur Folge. Man könnte natürlich zur Autovektorisierungsfähigkeiten moderner Compiler greifen (LLVM/CLang, GCC, ICC, MSVC), die nicht immer und nicht alle Konstrukte erkennt und (mit Ausnahme der Intel Compilier Collection) auch nicht zur Laufzeit dispatched werden kann. Warum nicht ICC? Weil bei der Intel Compiler Collection AMD Prozessoren nicht als “GenuineIntel” erkannt werden und für sie wird der langsamste Pfad ausgewählt. Zudem gilt der emittierte Maschinencode nur für CPU’s.

Im Groben unterscheidet sich die Entwicklung mit diesem Framework nicht von der Entwicklung mit purem C++, mit einer Außnahme:

- Das Framework ist auf größere Datenmengen ausgelegt und bietet entsprechende Funktionen an, um Datenfelder abzuarbeiten. Bei der Zusammenrechnung von zwei einzelnen Zahlen profitiert man nicht.
- Bei der Entwicklung muss man sich an die Sprunglose Arithmetik halten.

- Die Üblichen Container sind i.d.R nicht ohne Weiteres anwendbar, da sie mit skalaren Werten arbeiten.
- Die entwickelten Funktionen sind über die Scheduling und Dispatch Funktionalität des Frameworks aufzurufen. Diese kümmert sich um die passende Zweig-Auswahl, Multithreading und Datenzugriffe.

Beispiel für eine Addition MIT "gorynych":

Dieser Block wird dann entsprechend in alle von "gorynych" unterstützen Prozessor-Fähigkeiten umgesetzt.

```
VECTORIZED vreal add(const vreal &a, const vreal &b)
{
  return a + b;
}
```

Beispiel für eine Addition OHNE "gorynych":

Skalar: 1x float32

```
float add(float a, float b)
{
  return a + b;
}
```

SSE: 4x float32

```
__m128 add(__m128 a, __m128 b)
{
  return _mm_add_ps(a, b);
}
```

AVX: 8x float32

```
__m256 add(__m256 a, __m256 b)
{
  return _mm256_add_ps(a, b);
}
```

AVX512: 16x float32

```
__m512 add(__m512 a, __m512 b)
{
  return _mm512_add_ps(a, b);
}
```

OpenCL:

```
__kernel void add (__global const float* a, __global const float* b, __global float* result,
{
   const int idx = get_global_id(0);

   if (idx < num)
      res[idx] = src_a[idx] + src_b[idx];
}
```

An diesem sehr einfachen Beispiel sieht man bereits, dass man ohne “Gorynych” für jeden Befehlssatz andere Funktionen/Befehle" oder sogar Programmiersprachen verwenden und denselben Algorithmus mehrmals umsetzen und warten muss, was dem DRY-Prinzip widerspricht.

Ist Zustand:

Aktuell unterstützt “gorynych” x86-64 CPU Befehlssätze wie

- x87
- SSE2
- SSE3
- SSSE3
- SSE4.1/SSE4.2
- FMA3/FMA4
- AVX1
- AVX2

Aktuell bietet “gorynych” folgende Basisfunktionen:

- arithmetische Operationen
- logische Operationen
- lineare Algebra
- Rundung
- Basisfunktionen wie Absolutwert, Minimum, Maximum, Quadratische Wurzel, etc..

Soll Zustand:

- Mit “gorynych” erstellte Projekte sollen auch auf hochparallelen, modernen Grafikprozessoren lauffähig sein, die konfigurierbare Shaderkerne unterstützen.
- Damit entwickelte Projekte sollen möglichst ohne Anpassungen lauffähig sein.

- Dies bedeutet, dass die vorhandene Schnttstelle für GPGPU implementiert werden soll.

Systemarchitektur im Überblick

- Befehlssatzzweige
- x87 Kompatibilitätsschicht
- SSE
- AVX
- *GPGPU* (zu implementieren)
- Mathematische Funktionen
- Grundfunktionen
- Lineare Algebra
- Plattformhelfer
- Systeminformation
- Scheduler
  - CPU Basis
  - *GPGPU Basis* (zu implementieren)
- Dispatcher (zu erweitern)
- Hilfsfunktionen
- Speicherverwaltung
  - CPU (aligned memory)
  - *GPGPU* (zu implementieren)
- Angepasste Kollektionen
  - CPU (aligned memory)
  - *GPGPU* (zu implementieren)
- Sonstiges
- Makros
- Konstantengenerierung

Funktionale Anforderungen:

- Es müssen alle aktuell unterstützten Funktioen auch für GPU's umgesetzt werden.

Nicht-funktionale Anforderungen

Benutzbarkeit

- Für den Endanwender des Frameworks muss dieses logisch schlüssig und nachvollziehbar sein
- Eine entsprechende Dokumentation wird erweitert bzw. erstellt.

Wartbarkeit:

- Die Frameworkarchitektur muss weiterhin modular bleiben.
- Refactoring soll bei Bedarf und nicht ausgeschöpftem Zeitbudget erfolgen.

Zuverlässigkeit:

- Die angebotene Funktionalität wird im TDD-Verfahren entwickelt und validiert.
- Für jede Funktion sind entsprechende Unit-Tests notwendig.
- Codegenerierung wird getestet.
- Ausführungsergebnisse und Ausführungszeit werden ebenfalls getestet.

Abnahmekriterien

1. Iteration - Codgenerierung ist für die geforderte Funktionalität valide.
2. Iteration - Der generierte Code ist lauffähig und valide.
3. Iteration - Das Projekt 'solowej' ist mit der angepassten SIMD-Abstraktionsschicht zum Teil lauffähig und valide.
4. Iteration - Komplexere "solowej" Module, die LUT's verwenden, sin ebenfalls lauffähig und valide.