Fast and Easy GPU Offloading for Computational Finance

Lukasz Mendakiewicz, Microsoft

Introduction

- Computational finance characteristics
  - Endless search for performance
  - Large amounts of data to process
  - Custom hardware
- Challenges
  - Ninja programming
  - Harder to debug and maintain
- The ideal
  - Hiding hardware and platform differences in programming language layer while exposing maximum performance in a portable and productive manner.

What is C++ AMP?

- Programming model for expressing data parallel algorithms
- Exploit heterogeneous systems using mainstream tools
- Just C++ code, consisting of language extensions and libraries

- Introduced by Microsoft in Visual Studio 2012.

- What C++ AMP gives you?
  - Productivity: Write C++ code that runs on heterogeneous systems.
  - Portability: Write code once and run on various hardware/platforms.
  - Performance: Write C++ code that accelerate massively.

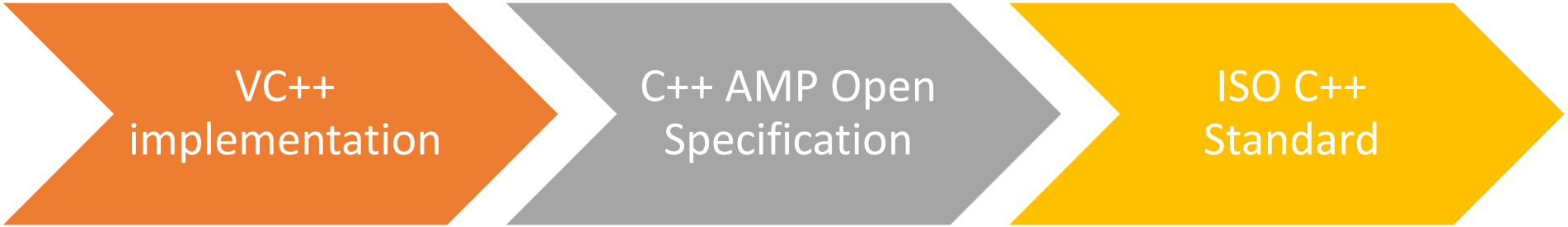
C++ AMP in computational finance

- Currently used by
  - CMA
    - To develop faster pricing mechanism
    - Proven in achieving competitive advantage
  - Frontline System
    - To model faster risk-based hedging and investment strategies
    - Proven in expanding the range of problems analyzed
- Case Studies
  - CMA has published case study detailing their use in pricing scheme
  - Frontline System case study detailing their use in linear and non-linear optimization solutions

> “By using C++ AMP, we can generate fast, accurate pricing with less strain on our resources, which is a key differentiator in our segment and helps us provide greater value to our clients”
>
> - Moody Hadi, Research Director, CMA

Standardization progress

- Open Specification
  - C++ AMP Open Specification v1.2 (Microsoft Community Promise)
  - C++ AMP Conformance Test Suite (Apache License 2.0)
- “Multidimensional bounds, index and array_view” ISO C++ proposal
  - Presented to LEWG in Issaquah, to be voted for Array TS in Rapperswil

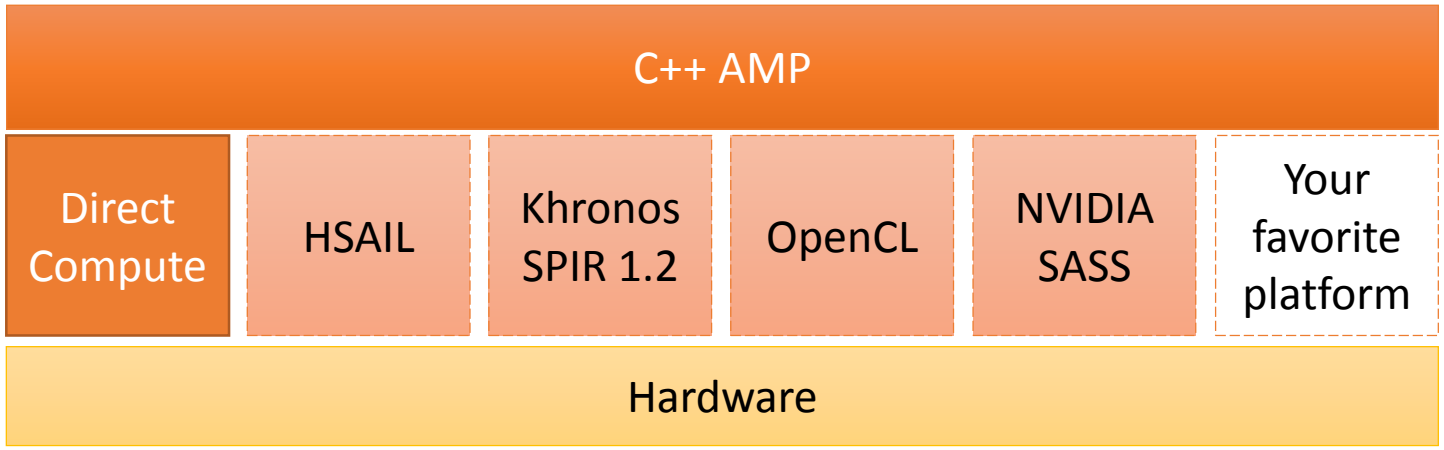


Open source community

- Clang/LLVM Compiler Support (NCSA License)
  - Targets OpenCL, SPIR and HSAIL
  - ETA for release: April 2014
- Open Source Libraries (Apache License 2.0)
  - AMD's BOLT Libraries
  - C++ AMP Algorithms Library (STL-style Algorithms)
  - C++ AMP RNG Library (Random Number Generator)
  - C++ AMP FFT Library (Fast Fourier Transform)
  - C++ AMP BLAS Library (Basic Linear Algebra Subroutines)
  - C++ AMP LAPACK Library (Linear Algebra Package)

C++ AMP as the high level language

- Offers consistent programming model across hardware and software platforms
- Supported by major compilers including…
  - Visual C++ compiler
  - Clang/LLVM
  - PathScale's ENZO, High performance compiler



MULTICORE WARE

AMD

HSA FOUNDATION

PathScale
High Performance Compilers

Platform support

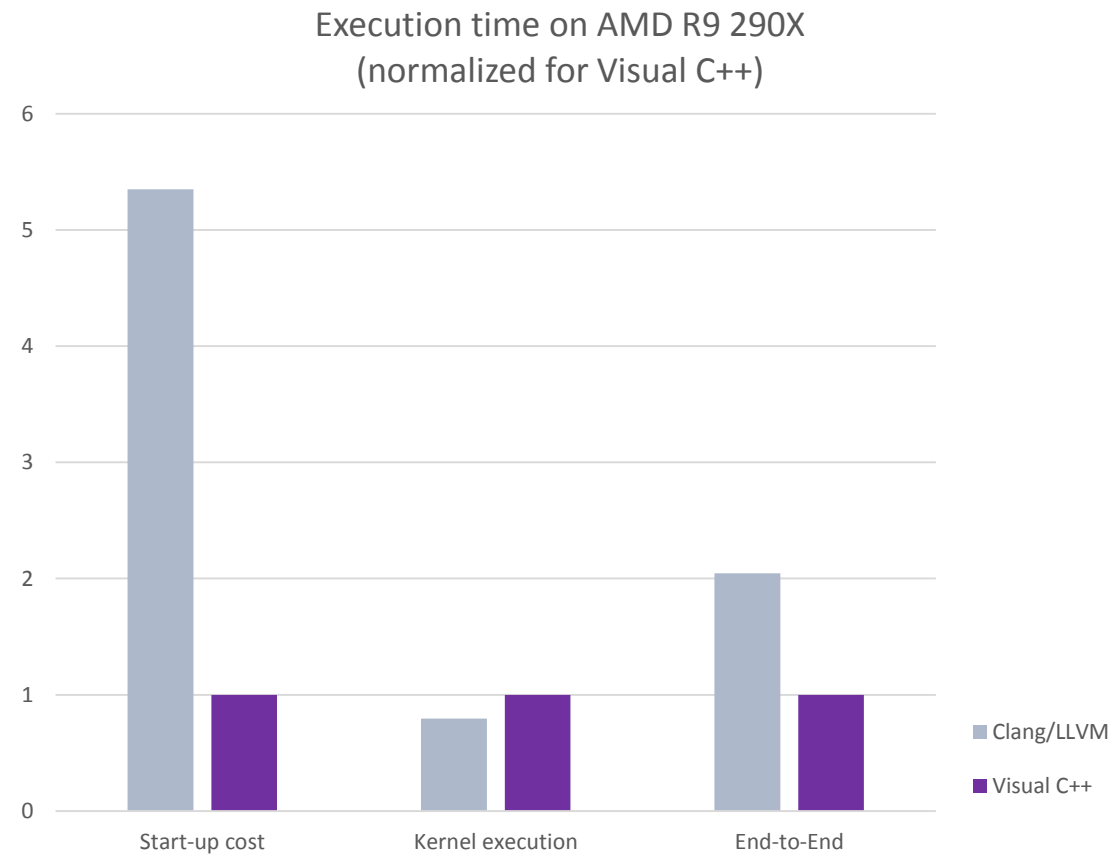
	Linux	Mac	Windows
AMD			Y
Intel			Y
NVIDIA			Y

Platform support

	Linux	Mac	Windows
AMD	Y	Y	Y
Intel	Y	Y	Y
NVIDIA	Y	Y	Y

Upcoming support on more platforms through HSA Runtime.

Performance benchmarks



Note: Start-up cost in Clang/LLVM needs to be fixed.

Microsoft road map for C++ AMP

- **Performance**
  - Leverage hardware and software evolution
- **Productivity**
  - Enhanced tooling support
  - Continue to invest in parallel algorithms
- **Portable**
  - ISO standardization of C++ AMP features
  - Foster expansion of C++ AMP ecosystem (compilers and libraries)

Q&A

Backup

Code introduction

SEQUENTIAL C++ CODE

```
#include <iostream>


int main()
{
    int v[11] = {'G', 'd', 'k', 'k', 'n', 31, 'v', 'n', 'q', 'k', 'c'};

    for (int idx = 0; idx < 11; idx++)
    {
        v[idx] += 1;
    }

    for(unsigned int i = 0; i < 11; i++)
        std::cout << static_cast<char>(  v[i]);
}
```

Code introduction

C++ AMP CODE

```
1.  #include <iostream>
2.  #include <amp.h>
3.  using namespace concurrency;

4.  int main()
5.  {
6.      int v[11] = {'G', 'd', 'k', 'k', 'n', 31, 'v', 'n', 'q', 'k', 'c'};

7.      array_view<int> av(11, v);
8.      parallel_for_each(av.extent, [=](index<1> idx) restrict(amp)
9.      {
10.         av[idx] += 1;
11.     });

12.     for(unsigned int i = 0; i < 11; i++)
13.         std::cout << static_cast<char>(av[i]);
14. }
```

Concept Count (5)

array_view: wraps the data to operate on the accelerator. array_view variables captured and associated data copied to accelerator (on demand)

parallel_for_each: execute the lambda on the accelerator once per thread

extent: the parallel loop bounds or computation “shape”

index: the thread ID that is running the lambda, used to index into data

restrict(amp): tells the compiler to check that code conforms to C++ subset, and tells compiler to target GPU

Links

https://bitbucket.org/multicoreware/cppamp-driver-ng

https://github.com/HSA-Libraries/Bolt

http://ampalgorithms.codeplex.com/

http://amprng.codeplex.com/

http://ampfft.codeplex.com/

http://ampblas.codeplex.com/

http://amplapack.codeplex.com/

http://blogs.msdn.com/b/nativeconcurrency/