AMD’s Unified CPU & GPU Processor Concept
Advanced Seminar Computer Engineering

Sven Nobis

Institute of Computer Engineering (ZITI)
University of Heidelberg

February 5, 2014
Overview

1. Introduction
2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA
3. Related Work
4. The way to HSA
   - Heterogeneous Unified Memory Access
5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools
6. Conclusion / Outlook
Previous: Single-Core Era

Single-Core Era

Enabled by:
- Moore’s Law
- Voltage Scaling

Constrained by:
- Power
- Complexity

Assembly ➔ C/C++ ➔ Java ...

Single-thread Performance

Time

[8, P. 5]
Today: Multi-Core Era

**Single-Core Era**

Enabled by:
- ✔ Moore’s Law
- ✔ Voltage Scaling

Constrained by:
- ✗ Power
- ✗ Complexity

**Multi-Core Era**

Enabled by:
- ✔ Moore’s Law
- ✔ SMP architecture
- ✔ Parallel SW
- ✔ Scalability

Constrained by:
- ✗ Power

**Assembly ➔ C/C++ ➔ Java …**

Throughput Performance

**pthreads ➔ OpenMP / TBB …**

Single-thread Performance

Time

Throughput Performance

Time ( nº of processors)

we are here

we are here

[8, P. 5]
Today till future: Heterogeneous System Era

---

Single-Core Era
- Enabled by:
  - Moore’s Law
  - Voltage Scaling
- Constrained by:
  - Power
  - Complexity

Multi-Core Era
- Enabled by:
  - Moore’s Law
  - SMP architecture
- Constrained by:
  - Power
  - Parallel SW
  - Scalability

Heterogeneous Systems Era
- Enabled by:
  - Abundant data parallelism
  - Power efficient GPUs
- Temporarily Constrained by:
  - Programming models
  - Comm. overhead

---

[8, P. 5]
Today’s problems on CPU / GPU programming
- programmability barrier
- communication costs

Solution
- AMD’s Unified CPU & GPU Processor Concept?
  → Heterogeneous System Architecture (HSA)
Introduction

- Today’s problems on CPU / GPU programming
  - programmability barrier
  - communication costs
- Solution
  - AMD’s Unified CPU & GPU Processor Concept?
    → Heterogeneous System Architecture (HSA)
Overview

1. Introduction

2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA

3. Related Work

4. The way to HSA
   - Heterogeneous Unified Memory Access

5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools

6. Conclusion / Outlook
CPU vs. GPU

CPU: LCU
Latency Compute Unit

GPU: TCU
Throughput Compute Unit
OpenCL & CUDA

- Both well-established platforms for GPU programming
- Compute Unified Device Architecture (CUDA)
  - Proprietary
  - Only for NVIDIA GPUs
- Open Computing Language (OpenCL)
  - Open standard
  - ATI, NVIDIA, Intel, ...
  - Not only GPUs
OpenCL Platform Model
OpenCL

Execution Model

[5, P. 11]
Overview

1. Introduction
2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA
3. Related Work
4. The way to HSA
   - Heterogeneous Unified Memory Access
5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools
6. Conclusion / Outlook
Related Work

- In CUDA [4]
  - Unified Virtual Addressing (UVA) in CUDA 4
  - Unified Memory in CUDA 6
  - Developer view to the memory
    - Implicit copy & pinning

- In OpenCL
  - Shared Virtual Memory

- Copy is still necessary (for fast access)
Overview

1. Introduction
2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA
3. Related Work
4. The way to HSA
   - Heterogeneous Unified Memory Access
5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools
6. Conclusion / Outlook
CPU and GPU cores in a single die

[3, P. 2] [7, P. 7]
Today: Non-Uniform Memory Access
- Different/partitioned physical memory per compute unit
- Multiple virtual memory address spaces

hUMA: Heterogeneous Unified Memory Access
- Same physical memory
- Same virtual memory for all compute units
Today: Non-Uniform Memory Access
- Different/partitioned physical memory per compute unit
- Multiple virtual memory address spaces

hUMA: Heterogeneous Unified Memory Access
- Same physical memory
- Same virtual memory for all compute units

[2, P. 7], [2, P. 8]
Required: hUMA Memory Controller

Features

- Shared page table support
  - Same large address space as the CPU
  - Page faulting
- Coherent memory regions
  - Fully coherent shared memory model
  - Like on today’s SMP CPU systems
Overview

1. Introduction

2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA

3. Related Work

4. The way to HSA
   - Heterogeneous Unified Memory Access

5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools

6. Conclusion / Outlook
Concepts

- Unified Address Space
  - Already mentioned with hUMA
- Unified Programming Model
- Queuing
- HSA Intermediate Language
Current programming models
  → Treating the GPU as a remote processor
  Extending existing concepts to use HSA
  - Programming languages like C++
  - *Task parallel* and *data parallel* APIs like C++ AMP
  - Stay in developers environment

```cpp
#include <iostream>
#include <amp.h>
using namespace concurrency;
int main() // "Hello World" in C++ AMP
{

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

    for(unsigned int i = 0; i < av.extent.size(); i++)
        std::cout << static_cast<char>(av(i));
}```
AMD’s Unified CPU & GPU Processor Concept

Sven Nobis

Introduction

Background

CPU vs. GPU
OpenCL & CUDA

Related Work

The way to HSA

Heterogeneous Unified Memory Access

HSA

Concepts
System Components
Development Tools

Conclusion / Outlook

References

[5, P.9]
Concepts

Queuing - New!

[5, P.9]
Concepts
HSA Intermediate Language

- HSAIL: HSA Intermediate Language
  - Bytecode
  - Designed for data parallel programming
  - GPU independent
- Generated by compilation stack \((later)\)
- Bytecode is compiled at runtime
  - to the *Hardware Instruction Set* of the current device
- Execution Model is similar to OpenCL
- **APU**
- **Software stack**
  - Compilation Stack
  - Runtime Stack
  - System (Kernel) Software
System Components
Compilation Stack

Compiler Front Ends
(OpenCL/C++AMP/other)

LLVM IR

Compiler Back End

HSAIL

[5, P. 15]
System Components
Runtime-Stack
Development Tools

- OpenCL
- C++ AMP: C++ Accelerated Massive Parallelism
- BOLT Library
- Aparapi
"HSA is an optimized platform architecture for OpenCL - Not an alternative to OpenCL" [8, P. 13]

- OpenCL on HSA will benefit from its features
Simple Example:

```cpp
#include <bolt/sort.h>
#include <vector>
#include <algorithm>

void main()
{
    // generate random data (on host)
    std::vector<int> a(1000000);
    std::generate(a.begin(), a.end(), rand);

    // sort, run on best device
    bolt::sort(a.begin(), a.end());
}
```

[9, P.5]
Development Tools
BOLT and C++ AMP

Simple Example:

```cpp
#include <bolt/transform.h>
#include <vector>

struct SaxpyFunctor
{
    float _a;
    SaxpyFunctor(float a) : _a(a) {};

    float operator() (const float &xx, const float &yy) restrict(cpu,amp)
    {
        return _a * xx + yy;
    }
};

void main()
{
    SaxpyFunctor s(100);
    std::vector<float> x(1000000); // initialization not shown
    std::vector<float> y(1000000); // initialization not shown
    std::vector<float> z(1000000);

    bolt::transform(x.begin(), x.end(), y.begin(), z.begin(), s);
}
```

[9, P.6]
Overview

1. Introduction

2. Background
   - CPU vs. GPU
   - Current Platforms: OpenCL & CUDA

3. Related Work

4. The way to HSA
   - Heterogeneous Unified Memory Access

5. Heterogeneous System Architecture
   - Concepts
   - System Components
   - Development Tools

6. Conclusion / Outlook
Conclusion

- Interesting concept
  - Simplifies development
  - Open up new possibilities
- Open platform
- In heavy development
  - Missing hardware with hUMA
    → Outlook
  - Software components not ready
→ A lot of potential

Conclusion / Outlook
Middle of January 2014:
- Kaveri APU is available [1]
- Desktop APU
- Support for
  - hUMA
  - Queuing
- Can connect both DDR3 and GDDR5 [11]

Server APU follows:
- Berlin
- ARM-Based: Seattle
References I


References II


References III

