Programming Model and Architecture Support for Efficient On-chip Heterogeneous Computing

Andrea Marongiu
L. Benini
Background

- Architectural heterogeneity and many-cores are THE design paradigm for embedded SoCs
- Witnessed by initiatives such as HSA, Khronos, and several products already architected in this manner
GPUs: massive data-parallelism for modest energy

- NVIDIA Tesla K40 discrete GPU: 4.3 TFLOPs, 235 Watts, $5,000

Integrated CPU+GPU processors

- More than 90% of processors shipping today include a GPU on die
- Low energy use is a key design goal

Intel 4th Generation Core Processor: “Haswell”

4-core GT2 Desktop: 35 W package
2-core GT2 Ultrabook: 11.5 W package

AMD Kaveri APU

4-core GT2 Desktop: 45-95 W package
Mobile, embedded: 15 W package

Heterogeneous SoC

Computing driven by general purpose programming + 3D gaming

Computing driven by multimedia, and low-power / limited resources
Heterogeneous SoC convergence

"APU", as replacement of traditional CPUs (e.g. SandyBridge, Fusion)
Compatible with legacy CPU, but optimally programmed with CUDA, OpenCL or DirectCompute

Lots of under-the-hood work to be done on the interconnect to avoid bottleneck on data transfers
H-SOC in 2013 – Apple A7

Used in IPad AIR & IPhone 5s
H-SOC in 2014(15) Tegra K1
Heterogeneous Computing in K1
Visual Analytics & Computational Photography
Target heterogeneous platform

ARM big-little Host system

Manycore cluster-based co-processor

Shared-memory CLUSTER

CCI-400 System NoC

Memory CTRL DDR2
The General-Purpose Programmable Accelerator (GPPA)

- Cluster-based platform, inspired by recent manycores
  - NVIDIA Fermi
  - Plurality HAL
  - Kalray MPPA
  - ST STHORM
OpenMP as a frontend

• Why OpenMP?
  – Widely adopted standard for shared memory parallel programming
    • Recently ported on various embedded MPSoCs
  – Shared memory model
    • Naturally fits our architecture
  – Simple directive-based parallelization interface

• Constructs for specifying parallelism
  – #pragma omp parallel

• Constructs for specifying work-sharing
  – #pragma omp for/sections/task

• Custom extensions to address specificities of target heterogeneous SoCs HW
Objectives

• Programmability issues:
  1. **NUMA** memory hierarchy in the GPPA
  2. Ease of coding for **heterogeneous** SoC
     – host+GPPA
     – On-cluster HW Processing Units (HWPU)
  3. Full-SoC **virtualization**

“[…] we foster [...] **OpenMP** as a means of simplifying programming of future accelerator-based heterogeneous MPSoCs by means of proper extensions to the basic APIs and the careful engineering of their implementation”
Objectives

- Programmability issues:
  1. **NUMA** memory hierarchy in the GPPA
  2. Ease of coding for **heterogeneous** SoC
     - host+GPPA
     - On-cluster HW Processing Units (HWPU)
  3. Full-SoC **virtualization**

“[…] we foster [...] **OpenMP** as a means of simplifying programming of future accelerator-based heterogeneous MPSoCs by means of **proper extensions to the basic APIs** and the **careful engineering of their implementation**”
NUMA communication

PE0 on CLUSTER 1 writes on data
Local communication
1 cycle latency

PE0 on CLUSTER 0 writes on data2
Remote communication
Increasing latency with distance
Nested Parallelism

while(1) {
    #pragma omp parallel num_threads(4)
    {
        #pragma sections
        {
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                ColorScaleConv();
            }
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                cvThreshold();
            }
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                cvMoments();
            }
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                cvAdd();
            }
        }
    }
}

A. void ColorScaleConv()
        {
            #pragma omp for
            for(i = 0; i < FRAME_SIZE; i++)
            {
                [ALGORITHM]
            }
        }

A powerful abstraction for specifying structured parallelism

And very suitable for NUMA (cluster-based) systems

But what about PHYSICAL thread mapping?
NUMA un-aware mapping
NUMA un-aware mapping

Cluster 0
- T0
- PE0
- T0
- PE3
- T1
- PE2
- T1
- PE1

Cluster 1
- PE0
- PE1
- PE2
- PE3

Cluster 2
- PE0
- PE1
- PE2
- PE3

Cluster 3
- PE0
- PE1
- PE2
- PE3

Diagram:
- T0
- T1
- T0
- T1
NUMA un-aware mapping

Cluster 0
- T0
- T0
- T0
- T0

Cluster 1
- T1
- T2
- T3
- T0

Cluster 2
- T2
- T3
- T1
- T2

Cluster 3
- T3
- T1
- T2
- T3

Diagram:
- T0
- T1
- T2
- T3

- T0
- T1
- T2
- T3

- T0
- T1
- T2
- T3

- T0
- T1
- T2
- T3

OpenMP Extensions

Proposed in the recent draft specification v4.0 (public review release candidate) Coupled to the `parallel` directive to specify thread mapping.

- **master**: assigns every thread to the same place as the master thread.
- **close**: assigns threads to places close to the place of the parent’s Thread.
- **spread**: creates a sparse distribution for a team of T threads among the P places of the parent’s place partition.
while(1) {
    #pragma omp parallel proc_bind (spread)
    {
        #pragma sections
        {
            #pragma section
            {
                #pragma omp parallel proc_bind (close)
                ColorScaleConv();
            }
            #pragma section
            {
                #pragma omp parallel proc_bind (close)
                cvThreshold();
            }
            #pragma section
            {
                #pragma omp parallel proc_bind (close)
                cvMoments();
            }
            #pragma section
            {
                #pragma omp parallel proc_bind (close)
                cvAdd();
            }
        }
    }
}

void ColorScaleConv()
{
    #pragma omp for
    for(i = 0; i < FRAME_SIZE; i++)
    {
        [ALGORITHM]
    }
}
Objectives (2)

• Programmability issues:
  1. NUMA memory hierarchy in the GPPA
  2. Ease of coding for **heterogeneous** SoC
     - On-cluster HW Processing Units (HWPU)
     - host+GPPA
  3. Full-SoC **virtualization**

“[… we foster […] OpenMP as a means of simplifying programming of future accelerator-based heterogeneous MPSoCs by means of proper extensions to the basic APIs and the careful engineering of their implementation ”
Tightly-Coupled, Shared-mem HW Processing Elements (HWPE)

- Shared memory–based control and data plane for HWPEs
  
  **Control plane**
  - How do processors offload computation to HWPEs?
  - How do processors synchronize with HWPEs?

  **Data plane**
  - How do HWPEs exchange data with processors?
Cores program HWPEs through memory mapped registers

A programming sequence requires:
1. addresses of inputs
2. addresses of outputs
3. a command to trigger execution

Synchronization via mem mapped register (polling) or HW events

HWPLUs access data directly from local TCDM through a master port
Objectives (2)

• Programmability issues:

1. **NUMA** memory hierarchy in the GPPA
2. Ease of coding for **heterogeneous** SoC
   - On-cluster HW Processing Units (HWPU)
   - host+GPPA
3. Full-SoC **virtualization**

“[… we foster […] **OpenMP** as a means of simplifying programming of future accelerator-based heterogeneous MPSoCs by means of **proper extensions to** the basic APIs and the **careful engineering of their implementation**”
Target heterogeneous platform

ARM big-little Host system

Manycore cluster-based co-processor

Can speak "virtual"

Host and GPPA can share VIRTUAL ADDRESS pointers

This GREATLY simplifies programming!

WHAT IF NO IOMMU IS AVAILABLE?

int a[20];
0x7a004

0x2a0c4

0xea0c8

PAGING
Supporting x86-64 Address Translation for 100s of GPU Lanes

Jason Power, Mark D. Hill, David A. Wood
Based on HPCA 20 paper
Summary

• CPUs & GPUs: physically integrated, logically separate
  
  **Near Future:** Cache coherence, *Shared virtual address space*

• Proof-of-concept GPU MMU design
  – Per-CU TLBs, highly-threaded PTW, page walk cache
  – Full x86-64 support
  – Modest performance decrease (2% vs. ideal MMU)

• Design alternatives not chosen
Motivation

• Closer physical integration of GPGPUs

• Programming model still decoupled
  – Separate address spaces
  – Unified virtual address (NVIDIA) simplifies code
  – **Want: Shared virtual address space** (HSA hUMA)
Separate Address Space

CPU address space

Simply copy data

Transform to new pointers

Transform to new pointers

GPU address space
Unified Virtual Addressing

CPU address space

1-to-1 addresses

GPU address space
void main() {
    int *h_in, *h_out;

    h_in = cudaHostMalloc(sizeof(int)*1024); // allocate input array on host
    h_in = ... // Initial host array

    h_out = cudaHostMalloc(sizeof(int)*1024); // allocate output array on host

    Kernel<<<1,1024>>>(h_in, h_out);

    ... h_out // continue host computation with result

    cudaHostFree(h_in); // Free memory on host
    cudaHostFree(h_out);
}
Shared Virtual Address Space

CPU address space

1-to-1 addresses

GPU address space
Shared Virtual Address Space

- No caveats

- Programs “just work”

- Same as multicore model
Shared Virtual Address Space

• Simplifies code
• Enables rich pointer-based datastructures
  – Trees, linked lists, etc.
• Enables composablity

Need: MMU (memory management unit) for the GPU
• Low overhead
• Support for CPU page tables (x86-64)
• 4 KB pages
• Page faults, TLB flushes, TLB shootdown, etc
Outline

• Motivation

• Data-driven GPU MMU design
  D1: Post-coalescer MMU
  D2: +Highly-threaded page table walker
  D3: +Shared page walk cache

• Alternative designs

• Conclusions
System Overview

- GPU
  - Compute Unit
  - Compute Unit
  - Compute Unit

- CPU
  - CPU Core
    - L1 Cache
    - L2 Cache

- DRAM
GPU Overview

Compute Unit (32 lanes)

Instruction Fetch / Decode

Register File

Coalescer

Scratchpad Memory

L1 Cache

GPU

Compute Unit

Compute Unit

Compute Unit

L2 Cache
Methodology

• **Target System**
  – Heterogeneous CPU-GPU system
    • 16 CUs, 32 lanes each
  – Linux OS 2.6.22.9

• **Simulation**
  – gem5-gpu – full system mode (gem5-gpu.cs.wisc.edu)

• **Ideal MMU**
  – Infinite caches
  – Minimal latency
Workloads

• Rodinia benchmarks
  – backprop
  – bfs: Breadth-first search
  – gaussian
  – hotspot
  – lud: LU-decomposition
  – nn: nearest-neighbor
  – nw: Needleman-Wunsch
  – pathfinder
  – srad: anisotropic diffusion

• Database sort
  – 10 byte keys, 90 byte payload
GPU MMU Design 0

Diagram of GPU MMU design with multiple CUs (Compute Units) each containing I-Fetch, Register File, TLB, Coalescer, and L1 Cache. There is also a L2 Cache connecting all the CUs.
GPU MMU Design 1

Per-CU MMUs:
Reducing the translation request rate
Scratchpad Memory

1x

0.45x
Reducing translation request rate

Breakdown of memory operations

- Shared memory and coalescer effectively filter global memory accesses
- Average 39 TLB accesses per 1000 cycles for 32 lane CU
GPU MMU Design 1

![Diagram of GPU MMU Design]
Performance

![Bar chart showing performance relative to ideal MMU for various tasks.]
GPU MMU Design 2

Highly-threaded page table walker: Increasing TLB miss bandwidth
Multiple Outstanding Page Walks

- Many workloads are bursty
- Miss latency skyrockets due to queuing delays if blocking page walker
Highly-threaded PTW

Per-CU TLBs:

Page walk buffers

<table>
<thead>
<tr>
<th>Outstanding addr</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outstanding addr</td>
<td>State</td>
</tr>
<tr>
<td>Outstanding addr</td>
<td>State</td>
</tr>
<tr>
<td>Outstanding addr</td>
<td>State</td>
</tr>
<tr>
<td>Outstanding addr</td>
<td>State</td>
</tr>
</tbody>
</table>

Page walk state machine

32

From memory

To memory
Performance

![Bar chart showing performance relative to ideal MMU for different designs. The chart compares Design 1 and Design 2 for various benchmarks such as backprop, bfs, gaussian, hotspot, lud, nn, nw, pathfinder, srad, sort, and average.]
WHAT IS THE HSA FOUNDATION

- HSA Foundation is a not for profit - industry standards body to create software/hardware standards for heterogeneous computing
  - simplify the programming environment
  - make compute at low power pervasive
  - introduce new capabilities in modern computing devices
- Core founders AMD, ARM, Imagination Technology, MediaTek, Qualcomm, Samsung, and Texas Instruments.
- Open membership to deliver royalty free specifications, and API’s
- Founded June 12, 2012
# HSA FOUNDATION BENEFITS

<table>
<thead>
<tr>
<th>Category</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Semiconductor</strong></td>
<td>• Neutral platform governance gives vendors the opportunity to influence heterogeneous architecture standards</td>
</tr>
<tr>
<td></td>
<td>• Ability to lower development cost for critical runtime foundations</td>
</tr>
<tr>
<td></td>
<td>• Technical sustainability of HSA via close alignment with key industry initiatives</td>
</tr>
<tr>
<td></td>
<td>• Diverse application ecosystem</td>
</tr>
<tr>
<td><strong>Platform &amp; OS Vendors</strong></td>
<td>• Commercial sustainability via multiple semiconductor members’ support</td>
</tr>
<tr>
<td></td>
<td>• Foundation that opens up innovative solutions to drive differentiation</td>
</tr>
<tr>
<td></td>
<td>• Diverse application ecosystem</td>
</tr>
<tr>
<td><strong>Device Manufacturers</strong></td>
<td>• Commercial sustainability via multiple semiconductor members’ support</td>
</tr>
<tr>
<td></td>
<td>• Foundation that opens up innovative solutions to drive differentiation</td>
</tr>
<tr>
<td></td>
<td>• Strong platform &amp; OS support</td>
</tr>
<tr>
<td><strong>ISVs &amp; Developers</strong></td>
<td>• Programming environment for advanced innovation</td>
</tr>
<tr>
<td></td>
<td>• Large addressable market</td>
</tr>
<tr>
<td></td>
<td>• Diverse routes to market</td>
</tr>
<tr>
<td></td>
<td>• Ability to contribute to HSA future in verticals of interest</td>
</tr>
<tr>
<td></td>
<td>• Commercial sustainability via strong commitments of HSA members</td>
</tr>
</tbody>
</table>
HSA FOUNDATION’S INITIAL FOCUS

Working to Attract Mainstream programmers
- Support broader set of languages beyond traditional GP-GPU languages
- Support for Task Parallel Runtimes & Nested Data Parallel programs
- Rich debugging and performance analysis support

Bring the GPU forward as a first class processor
- Unified coherent address space
- User mode dispatch/scheduling
- Can utilize pagable system memory
- Fully Coherent memory between the CPU and GPU
- Pre-emption and context switching
- Relaxed consistency memory model
- Quality of Service
Royalty Free IP, Specifications and API’s.

Two primary specifications are
- HSA Platform System Architecture Specification
  - Focus on hardware requirements and low level system software
  - Support Small Mode (32bit) and large mode (64bit)
- HSA Programmer Reference Manuel
  - Description HSAIL Virtual ISA
  - Binary format
  - Compiler Writers guide and Libraries developer guide
**AMD’S OPEN SOURCE COMMITMENT TO HSA**

- We will open source our linux execution and compilation stack
  - Jump start the ecosystem
  - Allow a single shared implementation where appropriate
  - Enable university research in all areas

<table>
<thead>
<tr>
<th>Component Name</th>
<th>AMD Specific</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSA Bolt Library</td>
<td>No</td>
<td>Enable understanding and debug</td>
</tr>
<tr>
<td>HSAIL Code Generator</td>
<td>No</td>
<td>Enable research</td>
</tr>
<tr>
<td>LLVM Contributions</td>
<td>No</td>
<td>Industry and academic collaboration</td>
</tr>
<tr>
<td>HSA Assembler</td>
<td>No</td>
<td>Enable understanding and debug</td>
</tr>
<tr>
<td>HSA Runtime</td>
<td>No</td>
<td>Standardize on a single runtime</td>
</tr>
<tr>
<td>HSA Finalizer</td>
<td>Yes</td>
<td>Enable research and debug</td>
</tr>
</tbody>
</table>
HSA
GREGORY STONER
WHAT ARE THE PROBLEMS WE ARE TRYING TO SOLVE

- The SOC are quickly following into the same many CPU core bottlenecks of the PC.
  - To move beyond this we need to look at right processor(s) and/or execution device for given workload at reasonable power

- While addressing the core issues of
  - Easier to program
  - Easier to optimize
  - Easier to load balance
  - High performance
  - Lower power
HSA TAKING PLATFORM TO PROGRAMMERS

- Balance between CPU and GPU for performance and power efficiency

- Make GPUs accessible to wider audience of programmers
  - Programming models close to today’s CPU programming models
  - Enabling more advanced language features on GPU
  - Shared virtual memory enables complex pointer-containing data structures (lists, trees, etc.) and hence more applications on GPU
  - Kernel can enqueue work to any other device in the system (e.g. GPU->GPU, GPU->CPU)
    - Enabling task-graph style algorithms, Ray-Tracing, etc.

- Clearly defined HSA memory model enables effective reasoning for parallel programming

- HSA provides a compatible architecture across a wide range of programming models and HW implementations.
HSA IS DESIGNED TO GO BEYOND THE GPU

- CPU
- SM&C
- GPU

Additional Components:
- Audio Processor
- Video Hardware
- Security Processor
- Fixed Function Accelerator
- DSP
- Image Signal Processing
- Shared Memory and Coherency
SIMPLIFIED HSA SOLUTION STACK

Application SW
- Domain Specific Libs (Bolt, OpenCV™, ... many others)
- Rendscript /OpenCl Runtimes
- OpenGL-ES Runtime
- Other Runtime

HSA Software
- HSA Runtime
- HSA Finalizer
- Kernal Driver

Drivers
- Legacy Driver
- Legacy Driver

Differentiated HW
- CPU(s)
- GPU(s)
- Other Accelerators

Application
Questions?

Thank You! ☺