Architecture and Software for Emerging Low-Power Systems

Wen-mei Hwu
Professor and Sanders-AMD Chair, ECE, NCSA, CS
University of Illinois at Urbana-Champaign
with
Jinjun Xiong (IBM), Nam Sung Kim, Deming Chen,
Izzat El Hajj, Abdul Dakkak, Liwen Chang, Simon Garcia, and Carl Pearson
Agenda

• Power-efficient computing in the past decade
• Revolutionary paradigm shift in applications
• Thoughts on the power-efficient computing in the next decade
Power-efficient Computing in the Past Decade

- Giving up a small level of performance saves a lot of power

- For single thread
  - Clock frequency
  - Operation latencies
CPUs: Latency Oriented Design

- High clock frequency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
- Powerful ALU
  - Reduced operation latency
GPUs: Throughput Oriented Design

- Moderate clock frequency
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies
Applications Benefit from Both CPU and GPU

- CPUs for sequential parts where latency matters
  - CPUs can be 10+X faster than GPUs for sequential code

- GPUs for parallel parts where throughput wins
  - GPUs can be 10+X faster than CPUs for parallel code

Amdahl’s Law is alive and well!
Blue Waters Computing System
Operational at Illinois since 3/2013

- 49,504 CPUs -- 4,224 GPUs
- 12.5 PF
- 1.6 PB DRAM
- $250M
- Sonexion: 26 PBs
- Spectra Logic: 300 PBs
- IB Switch: >1 TB/sec
- 10/40/100 Gb Ethernet Switch: 100 GB/sec
- WAN: 120+ Gb/sec
Current State of Heterogeneous Computing

- Throughput computing using GPUs have resulted in 2-5X end-to-end application-level performance improvement
  - A ten-year journey – first CUDA GPUs (G80) came out in 2007

- GPUs, big data and deep learning have formed a positive spiral for the industry

- GPU computing has so far had narrow but deep impact on applications (C-FAR)
  - Data movement overhead and small GPU memory
    - Unified memory, HBM, NVLink, and HSA-style systems

  - Low-level programming interfaces with poor performance portability
    - C++/Python style programming and software synthesis systems
Hierarchical Compute Organization of Devices

**CPU**
1. Process
2. Thread (vector-capable)
3. Vector Lane
4. Instruction-level Parallelism

**GPU**
1. Grid
2. Block
3. Warp
4. Thread
5. Instruction-level Parallelism
Hierarchical Programming of CPUs

1. Process
2. Thread (vector-capable)
3. Vector Lane
4. Instruction-level Parallelism

```c
nt = omp_get_num_threads();
tile = (len + nt - 1)/nt;
#pragma omp parallel
{
    j = omp_get_thread_num();
    accum = 0;
    #pragma unroll
    for(int i = 0; i < tile; ++i) {
        accum += in[j*tile + i];
    }
    partial[j] = accum;
}
sum = 0;
for(int j = 0; j < nt; ++j) {
    sum += partial[j];
}
return sum;
```
Hierarchical Programming of GPUs

1. Grid
2. Block
3. Warp
4. Thread
5. Instruction-level Parallelism

```c
int tile = (len + blockDim.x - 1)/gridDim.x;
int sub_tile = (tile + blockDim.x - 1)/blockDim.x;
int accum = 0;

#pragma unroll
for(unsigned i = 0; i < sub_tile; ++i) {
    accum += in[blockIdx.x*tile + i*blockDim.x + threadIdx.x];
}

int tmp[threadIdx.x] = accum;
__syncthreads();
for(unsigned s=1; s<blockDim.x; s *= 2) {
    if(id >= s)
        tmp[threadIdx.x] +=
            tmp[threadIdx.x - s];
    __syncthreads();
}

int partial[blockIdx.x] = tmp[blockDim.x-1];
return; // Launch new kernel to sum up partial
```
Tangram: Codelet-based Programming Model

```cpp
codelet
int sum(const Array<1,int> in) {
    unsigned len = in.size();
    int accum = 0;
    for(unsigned i=0; i < len; ++i) {
        accum += in[i];
    }
    return accum;
}

(a) Atomic autonomous codelet

codelet coop __tag(kog)
int sum(const Array<1,int> in) {
    __shared int tmp[coopDim()];
    unsigned len = in.size();
    unsigned id = coopIdx();
    tmp[id] = (id < len) ? in[id] : 0;
    for(unsigned s=1; s < coopDim(); s *= 2) {
        if(id >= s)
            tmp[id] += tmp[id - s];
    }
    return tmp[coopDim()-1];
}

(b) Atomic cooperative codelet

codelet __tag(asso_tiled)
int sum(const Array<1,int> in) {
    tunable unsigned p;
    unsigned len = in.size();
    unsigned tile = (len+p-1)/p;
    return sum( map( sum, partition(in, p,sequence(0,tile,len),sequence(1),sequence(tile,tile,len+1))));
}

(c) Compound codelet using adjacent tiling

codelet __tag(stride_tiled)
int sum(const Array<1,int> in) {
    tunable unsigned p;
    unsigned len = in.size();
    unsigned tile = (len+p-1)/p;
    return sum( map( sum, partition(in, p,sequence(0,1,p),sequence(p),sequence((p-1)*tile,1,len+1))));
}

(d) Compound codelet using strided tiling
```
Agenda

• Power-efficient computing in the past decade
• Revolutionary paradigm shift in applications
• Thoughts on the power-efficient computing in the next decade
# A Revolutionary Paradigm Shift

<table>
<thead>
<tr>
<th>20&lt;sup&gt;th&lt;/sup&gt; Century</th>
<th>21&lt;sup&gt;st&lt;/sup&gt; Century</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small mask patterns</td>
<td>Optical proximity correction</td>
</tr>
<tr>
<td>Electronic microscope and Crystallography with computational image processing</td>
<td>Computational microscope with initial conditions from Crystallography</td>
</tr>
<tr>
<td>Anatomic imaging with computational image processing</td>
<td>Metabolic imaging measures disease progress before visible anatomic change</td>
</tr>
<tr>
<td>Teleconference</td>
<td>Cognitive Immersive Room</td>
</tr>
<tr>
<td>GPS</td>
<td>Self-driving cars</td>
</tr>
</tbody>
</table>
How are applications changing?

• Applications that use large, accurate first-principle-based models
  • Problems that we know how to solve accurately but choose not to because it would be “too expensive”
  • High-valued applications with approximations that cause inaccuracies and lost opportunities
  • Medicate imaging, remote sensing, earthquake modeling, weather modeling, astrophysics modeling, precision digital manufacturing, combustion modeling, ....

• Applications that we have failed to program
  • Problems that we just don’t know how to solve
  • High-valued applications with no effective computational methods
  • Computer vision, natural language dialogs, document compression, stock trading, fraud detection, ...
Agenda

• Power-efficient computing in the past decade
• Revolutionary paradigm shift in applications
• Thoughts on the power-efficient computing in the next decade
IBM-Illinois C³SR faculties & students (Est. 9/2016)
Wen-mei Hwu (Illinois) and Jinjun-Xiong (IBM) Co-directors

Suma Bhat  Minh Do  Deming Chen  Julia Hockenmaier  Wen-mei Hwu  Nam Sung Kim  Dan Roth  Rakesh Nagi  Lav Varshney
The Three Pillars of C3SR:

• Creative experiential learning advisor (CELA) as a grand challenge use case for cognitive capabilities

• Cognitive application builder (CAB) to make the underlying heterogeneous infrastructure easy to consume for cognitive application developers

• Cognitive systems innovations (Erudite) for workload acceleration, including Near Memory Acceleration (NMA)
Problem Statement for the Erudite Project

• Latency and bandwidth limitations on accessing massive data sets
  • Sweeping through large data sets brings systems to their knees
  • Low data reuse creates unnecessary traffic through the memory hierarchy
  • Sustained performance < 1% of peak for memory bound applications

• Large software overhead for data access
  • File system overhead and bottleneck
  • Message passing serialization/deserialization and layers of constructors
  • Excessive data copying between memory address spaces and subspaces
A Simplified Heterogeneous System
(IBM Minsky with NVIDIA Pascal GPUs)

- CPU Host (~1 TFLOPS)
- DDR Memory System (~100 GBs)
- Storage (~10 TBs)
- GPU 1 (~10 TFLOPS)
- GPU 2 (~10 TFLOPS)
- HBM (~10 GBs)
Ideal GPU Computation Patterns

- Each memory operand should be reused ~50 times among threads and thread blocks
- Data should fit into GPU Memory
  - Or reused many times when transferred to the GPU memory
- Data should be accessed with spatial locality
Example: Direct vs. Iterative Solvers

**Direct Solvers**
- Good Locality
  - Data reuse through tiling
- Sparsity
  - Too many fill-ins, data explosion
- Stability
  - Pivoting restricts parallelism

**Iterative Solvers**
- Poor Locality
  - Multiple sweeps through matrix
- Good with Sparsity
  - No fill-ins during solution time
- Stability
  - Convergence varies
  - Preconditioning may enlarge matrix
Iterative Solver Example – If matrix fits into GPU Memory

- CPU Host (~1 TFLOPS)
  - DDR Memory System (~100 GBs)
    - Storage (~10 TBs)
  - HBM (~10 GBs)

- GPU 1 (~10 TFLOPS)
  - HBM (~10 GBs)
  - 80 GB/s
  - 700 GB/s

- GPU 2 (~10 TFLOPS)
  - HBM (~10 GBs)
  - 80 GB/s
  - 700 GB/s

- ~100 GFLOPS Sustained

ECE ILLINOIS
Iterative Solver Example – If matrix fits into Host Memory

~10 GFLOPS
Sustained

CPU Host (~1 TFLOPS)

GPU 1 (~10 TFLOPS)

GPU 2 (~10 TFLOPS)

Tremendous loss of both performance and energy efficiency
Iterative Solver Example – If matrix has to be accessed from storage

< 1 GFLOPS Sustained

CPU Host
(~1 TFLOPS)

Storage
(~10 TBs)

100 GB/s

80 GB/s

80 GB/s

10 GB/s

DDR Memory System
(~100 GBs)

GPU 1
(~10 TFLOPS)

GPU 2
(~10 TFLOPS)

HBM
(~10 GBs)

HBM
(~10 GBs)

700 GB/s

700 GB/s

ECE ILLINOIS
Erudite Project Goals

• To achieve > 100x performance/Watt for data-intensive cognitive computing applications
  • Collaborative heterogeneous execution of CPU, GPU, and NMA
  • Elimination of file-system software overhead for engaging large data sets
  • Placement of computation appropriately in the memory and storage hierarchy

• To drastically improve the efficiency of distributed cognitive software architectures
  • Highly optimized kernel synthesis for heterogeneous computing components
  • Optimized scheduling of workflows for improved performance and efficiency
  • Elimination of software serialization/deserialization overhead in communication
  • Elimination of software overhead in processing traditional network messages
Erudite Step 1: remove file system from data access path

- CPU Host (~1 TFLOPS)
- DDR/Flash Memory System (~10 TBs) 100 GB/s
- GPU 1 (~10 TFLOPS) 700 GB/s
- GPU 2 (~10 TFLOPS) 700 GB/s
- HBM (~10 GBs)
- Storage (~10 TBs) 10 GB/s
Erudite Step 2: place NMA compute inside memory system

DDR/Flash Memory System (~10 TBs)

100+ GFLOPS NMA Compute Proportional to data capacity

CPU Host (~1 TFLOPS)

GPU 1 (~10 TFLOPS)

GPU 2 (~10 TFLOPS)

HBM (~10 GBs)

100 GB/s

80 GB/s

700 GB/s

80 GB/s

700 GB/s
Erudite Memory Board 1.0

- Develop a principled methodology for acceleration
  - HLS (high-level synthesis) for FPGA based on FCUDA and TANGRAM
  - Hardware/Software partitioning for heterogeneous systems
  - Optimized for cognitive workload
Erudite Step 3

collaborative heterogeneous computing (Chai)

![Program Structure](image)

- **Data-parallel tasks**
- **Sequential sub-tasks**
- **Coarse-grained synchronization**

---

**Fine-grained Task Partitioning**

Device 1

Device 2
Erudite Target Computation Types

• Large iterative solvers
  • Equations, constraints, etc.

• Low-complexity solver algorithms
  • Multi-level Fast Multipole Methods, etc.

• Graph analytics
  • Inference, search, counting, etc.

• …
Longer Term Power-Efficiency Agenda

• Package-level integration
  • Post-Moore scaling
  • Optical interconnects in package?
  • Collaboration support for heterogeneous devices

• System software (r)evolution
  • Persistent objects for multi-language environments
  • Directory and mapping of very large persistent objects

• Power consumption in memory
  • Much higher memory-level parallelism needed for SSD-based main-memories
  • Latency vs. throughput oriented memories

![Energy Efficiency vs. Distance Graph](image)
Conclusion and Outlook

• Heterogeneous computing for power-efficiency
  • Latency vs. throughput compute designs have empowered application paradigm shift
  • Latency vs. throughput memory designs will be the next focus

• Drivers for low-power systems
  • Large-scale inverse problems with natural data inputs
  • Machine-learning-based applications

• Erudite cognitive computing systems project
  • Removing file-system bottleneck from access paths to large data sets
  • Placing compute into the appropriate levels of the memory system hierarchy
  • Memory parallelism (data bandwidth) proportional to the data capacity
  • 100x improvement in the power-efficiency in emerging applications
Thank you!