What every computer scientists should know about computer architects

Henri-Pierre Charles
Bastien GIRAUD, Jean-Philippe NOËL, Maha KOOLI, Clément TOUZET

CEA DACLE department / Grenoble

21/11/2017
1 Introduction

2 Dynamic Code Generation

3 In Memory Computing
CEA Structure

Each structure has different

- research objectives
- funding models
- the important word: “Commissariat”
- CEA-DRT
  - Collaborative projects
  - Industrial partnership
Introduction : Motivation

About title joke
- “What Every Computer Scientist Should Know About Floating-Point Arithmetic” 1991 David Goldberg
- “What Every Programmer Should Know About Memory” November 21, 2007 Ulrich Drepper

Warning!
- Computer science is about “multidisciplinarity”

Disciplinarity in computer science
(Alphabetic order, no offense ;-)
- Algorithms
- Applied math
- Code optimization
- Memory design
- Network
- Processor design
- Theory
- ../.. (put your preferred science classification here)
Introduction: From Application to Binary Code

Programming Language Semantic Bottleneck

High level optimization

1. Application Domain
   - Floating point computation
   - Telecommunications
   - Integer computation
   - Physic simulation
   - Linear algebra
   - Cryptography
   - 2D graphic
   - 3D graphic

2. Hardware
   - Integer arithmetic
   - Floating point arithmetic
   - Saturated arithmetic
   - Complex arithmetic
   - Vector arithmetic
   - Matrix arithmetic
   - Bit level shuffling

3. Compiler
   - Integer arithmetic
   - Floating point arithmetic
   - Vector arithmetic
   - Matrix arithmetic

4. Programming
   - Floating point arithmetic
   - Integer arithmetic

Algorithmic -> Programming -> Compilation -> Execution
“Parallel View from Berkeley”

Figure 2. Bridge analogy connecting users to a parallel IT industry, inspired by the view of the Golden Gate Bridge from Berkeley, CA.
Ultra-Wide Body-Bias Range LDPC Decoder in 28nm UTBB FDSOI Technology

Introduction: Design Example

ISSCC 2013 / SESSION 24 / ENERGY-AWARE DIGITAL DESIGN / 24.3

24.3 Ultra-Wide Body-Bias Range LDPC Decoder in 28nm UTBB FDSOI Technology

Philipp Terrel, et al.

Introduction

There are 3 instances of the LDPC decoder in the chip (the chip micrograph in Fig. 24.3.1). The decoder assigns the 415 nodes required by the 802.11n standard (see the ITR1 [5]) to 512 levels. Each level contains a 1124 nodes, corresponding to 512 levels, and the number of mistakes in each level is limited to 1124. This number is then used to determine the best tradeoff in delay and energy for a given operating point. We are able to decrease minimum bulk EDP (1V/0.3V) by 43% with LPDC_FW, which is 2× leakage reduction compared to LPDC_BW, while bulk only achieves a leakage 2× reduction. On the other hand, a performance-oriented design like LPDC_FW is able to achieve a 2× leakage reduction, with a lower absolute value than bulk.

Fig. 24.3.4 shows normalized leakage power vs. Vbb at VDD=0.6V and at 25°C. We are able to achieve a 4× reduction in total power vs. bulk. Similarly, at constant power dissipation, the frequency achieved by the UTBB design is 35% higher than the bulk design. As a result, a 49% reduction in total power vs. bulk is achieved. Furthermore, when no BB is applied compared to a bulk FBB of 300mV. Furthermore, an additional 5× reduction in total power is achieved with LPDC_FW, which is 2× leakage reduction compared to LPDC_BW, while bulk only achieves a leakage 2× reduction. On the other hand, a performance-oriented design like LPDC_FW is able to achieve a 2× leakage reduction, with a lower absolute value than bulk.

On the other hand, a performance-oriented design like LPDC_FW is able to achieve a 2× leakage reduction, with a lower absolute value than bulk.

In this paper, we report on a 28nm UTBB VLSI circuit that is fully functional in 28nm UTBB. The chip implements an 802.11n LDPC decoder that can decode 802.11n LDPC codes with data rates of up to 720MB/s. The chip was fabricated in a 28nm UTBB process and was characterized in a 1.5V supply and 25°C environment. The chip is 3.5mm x 4.5mm and consumes 89.4mW at 1.5V supply, which corresponds to 1.5V supply and 25°C environment. The chip is 3.5mm x 4.5mm and consumes 89.4mW at 1.5V supply, which corresponds to 1.5V supply and 25°C environment. The chip is 3.5mm x 4.5mm and consumes 89.4mW at 1.5V supply, which corresponds to 1.5V supply and 25°C environment.

Acknowledgments

The authors would like to thank the CCDS and STD teams from UTBB FDSOI Technology.
**Introduction**

**Bottom up description**
- Technology characterization / simulation (atomistic level simulation) BigDFT, TBSIM
- Gate level characterization / simulation (gate level) Spice level
- Architectural level (RTL, SystemC, ..)
- Application level (GEM5, QEMU, LLVM) + power consumption model (MCPAT, ..)

**Varying Research objective**
- Simulate many physical device, old simulation code
- Show / validate an physical concept, jump to a new one
- Show / validate an architectural concept, jump to a new one
- Gain 0.1 % on SpecInt or SpecFloat on old technology
Intro : Compilers

Classical Compiler architecture: GCC, CLANG
- Driven by performance only
- Mono architecture
- Not energy aware

Before / After
Future Compilers Architecture

- Multi objective (execution time, power, thermal constraints)
- Multi-target (heterogeneous multi SoC)
- Data driven (dynamically)

Before / After
deGoal Flow

Compilation flow

- .cdg
  - C source
  - degoal high-level ASM
- .C
- static binary
  - compilette
- runtime binary
  - compilette
  - kernel

developer
degaultoc
C compiler
compilette
RUN TIME (data adaptation)

REWRITE TIME (source to source)
STATIC COMPILATION TIME
HW desc.
data
Example: Simple multiplication

Pseudo constant multiplication

```c
#include <stdio.h> /* -*- c -*- */

typedef int (*pifi)(int);

/* Compilette which add a constant value */
pifi multiplyCompile(int multiplyValue)
{
    cdgInsnT *code = CDGALLOC(1024);
    printf("Code\n
generation\nfor\nmultiply\nvalue\n%d\nc\nat\np\n", multiplyValue, code);
    
    VectorType ScalarInt float 32 1
    RegAlloc ScalarInt in 1

    Begin code Prelude in0

    mul in0, in0, #(multiplyValue)
    rtn
    End

}#
return (pifi)code;
```
Use Case: Performance

Results
- Each point represents a performance in the data space
- Dongara’s MAGMA versus compilette version

Illustration
(a) SGEMM MAGMA
(b) SGEMM on-line search

Performances
- Simpler code, tiling adapted to run time data
- CPU & GPU runtime selection & adaptation
In memory computing : Global View

Research

- Power, power, power
- Density, density, density
- Heterogeneous architectures
  - MPSoC (Multi Processor System on Chip)
  - Same package
  - Multiple technologies

Global Idea

Team
Bastien GIRAUD, Jean-Philippe NOËL, Henri-Pierre CHARLES, Maha KOOLI, Clément TOUZET
Multiple possibilities

- MPSoC (Memory on chip)
- Multicore with shared / distributed memory (OpenMP, MPI)
- GPU (Cuda, OpenCL)
- Near data processing
- Logic in memory
- Processing in memory
- Computing in memory

IMPACT : In Memory Processing Power Aware Computing

Other approaches

Focused on architecture only (no SW), and on DRAM only (no NV)
**Impact : In Memory Ideas**

**Features**
- Pipelined operations
- Instructions send by processor
- No sequencer inside memory

**IMPACT priciple**

- Memory Bit-cell Array
- N-bytes
- M-rows
- Pipeline Depth
- Operation Flow

- OR
- ADD
- SET

- 1st Cycle OR
- 2nd Cycle OR
- 1st Cycle ADD
- 2nd Cycle ADD
- 3rd Cycle ADD
- 1st Cycle SET

---

**Introduction**

**Dynamic Code Generation**

**In Memory Computing**
Impact : Research Organisation

Research planning

- Basic Ideas / Initial ISA
- Algorithmic evaluation / memory design / sizing: high level simulation
- Test chip launch in 2018
- Refine ISA, Work on compilation
- Test chip characterization
- Test chip launch in 2019

Patents, publish, launch chips

Funding

- Internal CEA project (CARNOT project)
- Submitted ANR project
- .. more to come
**Impact: Domains Evaluation**

**Good for?**
- Logic (NOR / AND) at the bit cell level
- Computing (integer) at the line level
- At “memory line size” level parallelism
- Multiple Line Selection

**Scenarios Evaluated so far**
- Simple cryptography (OTP)
- Image processing
- Computing
- Database
How evaluate performance for a non existant memory circuit?

- No compiler
- No programming language
- No programming model
- No power estimation from real hardware

**Illustration**

Conventional System vs IMPACT System

**Programming language**

- Conventional: Bench (C) → Compilation (clang)
- IMPACT: Smart Bench (C)

**Middleware**

- Conventional: LLVM IR → Instrumentation
- IMPACT: Smart LLVM IR

**Emulation**

- Conventional: Instrumented LLVM IR → Execution (lli)
- IMPACT: Instrumented Smart LLVM IR

**Performance Evaluation**

- Conventional: Trace Analysis → Speed Factor
- IMPACT: Trace Analysis → Speed Factor

- IMPACT vs. Conventional

- Manual transformation (similar to OpenCL or CUDA programming model)
- Automatic transformation using our implemented tools
- Automatic generation using existing tools
Introduction
Dynamic Code Generation
In Memory Computing

Impact : Energy Model 1/3

Impact : Energy Model 1/3

Charaterization from an other platform

- MPSoC ARM / FDSOI 28nm
- Special connectors :
  - Memory energy
  - Core energy
- Instruction level energy measurement

Illustration
Energy model from real platform

- Extract memory & core energy from ARM cortex M0+
- Use these values for “conventional system”
- Use “core”, “fetch” and “memAccess” part for IMPACT system
- Inject energy values into trace applications

Illustration

(a) Energy Measurement for a set of Instructions

(b) Energy Modelling for ‘Sub A, B, C’ Operation

(A, B, C are of 2048-bits)

Impact : Energy Model 2/3
Experimental platform

- Patch LLVM intermediate representation
- Use LLVM vector operators
- Trace execution
- Evaluate execution traces with power model

Illustration

```c
typedef unsigned char Array;

__attribute__ ((ext_vector_type(N))));

void IMPACT()

Array A, B, C;
C = A + B

define void @IMPACT() {
    %2 = add <256 x i8> %0, %1
    ...
}

add 2048 0x7fb000 0x7fac00 0x7fc000
```

Execution time: 3 cycles
Energy: 141.36 pJ (ARM Cortex-M0+)

Impact: Energy-Model 3/3
**Initial results**

**Motion Detection**
- Energy Reduction
- Image Size (Memory Row- & Column-dependency)
- 6T bitcell
- 10T bitcell

**One-Time Pad**
- Energy Reduction
- Key Size (Only Memory Column-dependency)
- 6T bitcell
- 10T bitcell

**Boolean Matrix Multiplication**
- Energy Reduction
- Matrix Size (Memory Row- & Column-dependency)
- 6T bitcell
- 10T bitcell

**Data Base Update**
- Energy Reduction
- Number of DB Match/DB Size (Only Memory Row-dependency)
- 125/500
- 250/500
- 375/500
- 500/1000
- 750/1000
Impact First Dimensioning

Memory dimensioning

Execution Time Speed-Up / Energy Reduction Factor

- 1024-Memory Row
- ~250x/9x
- ~499x/10x
- ~2000x / 29x
- ~4100x / 29x
- ~9200x / 29x
- ~5760x/29x
- ~11008x / 29x

Memory Column Size (Kbit)
Conclusion

Already published

- “Software Platform Dedicated for In-Memory Computing Circuit Evaluation” Maha Kooli, Henri-Pierre Charles, Clément Touzet, Bastien Giraud and Jean-Philippe Noel, RSP 2017, ESWEEK
- Accepted DATE presentation
- more to come!

Need to do

- Refine instruction set architecture deduced from applications
- Find suitable programming model
- Optimize or compile?