AI Omnipresence

Speech & NLP

Computer Vision

Recommendation Systems
Computing AI

Cloud Computing (e.g. Data Centers)

- Great for extreme parallelization
- Computation and memory intensive workloads
Computing AI

Cloud Computing (e.g. Data Centers)
- Great for extreme parallelization
- Computation and memory intensive workloads

Edge Computing (e.g. smartphones, wearables, smart speakers)
- More energy efficient
- Lower latency
- Better privacy
Edge AI chips from our lab

Programmable DNN Classifier for IoT

Always-on processor for IoT DNN inference tasks

Processor for compute intensive kernels in DNN, DSP and security algorithms

Processor for unsupervised probabilistic and Bayesian perception tasks

Processor for speech and natural language processing tasks
Edge AI chips from our lab

Programmable DNN Classifier for IoT
Always-on processor for IoT DNN inference tasks
Processor for compute intensive kernels in DNN, DSP and security algorithms
Processor for unsupervised probabilistic and Bayesian perception tasks
Processor for speech and natural language processing tasks

How do you design them?
A brief history – The Transistor

• 1947: First transistor (Bell Labs)
A brief history – The Transistor

MOSFET: metal–oxide–semiconductor field-effect transistor
A brief history – The Transistor

• 1947: First transistor (Bell Labs)

• 1958: First integrated circuit (Kilby @ Texas Instruments)

• 1961: First useful IC in Si (Noyce @ Fairchild)

• 1971: Intel 4004 microprocessor

• 2007: Intel Core i7
Moore’s Law is slowing down

The number of transistors in a dense integrated circuit (IC) doubles about every two years.
Moore’s Law is slowing down

The number of transistors in a dense integrated circuit (IC) doubles about every two years.
Moore’s Law is slowing down

The number of transistors in a dense integrated circuit (IC) doubles about every two years.

[https://upload.wikimedia.org/wikipedia/commons/8/8b/Moore%27s_Law_Transistor_Count_1971-2018.png]
Dennard’s Law actually ended

“As the dimensions of a device go down, so does power consumption – keeping the power density constant”

Robert Dennard
Dennard’s Law actually ended

“\textit{As the dimensions of a device go down, so does power consumption – keeping the power density constant}”

original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010–2017 by K. Rupp
Dennard ignored the impact of leakage current and voltage limits.
Dennard’s Law explained

2x transistor reduction
Dennard’s Law explained

- 2x transistor reduction
- Same power consumption according to Dennard’s Law
Dennard’s Law explained

2x transistor reduction

Same power consumption according to Dennard’s Law

No longer the case since the mid-2000s
We are in the era of hardware specialization!
We are in the era of hardware specialization!

- Hardware Accelerators
- Specialized Architectures
- Domain-Specific Processors
- Application-Specific Integrated Circuits (ASICs)
Neural Networks

Key kernel: matrix multiplication
Matrix Multiplication: Example

Let’s assume B and C are both 100,000 x 100,000

 pseudo-code

for(int i =0; i < x; i++)
    for(int j =0; j < y; j++)
        for(int k=0; k < z; k++)
            A[i][j] += B[i][k]*C[k][j]
Matrix Multiplication: Example

Let’s assume B and C are both 100,000 x 100,000

Assuming this pseudo-code is used to do the matrix multiplication of B x C, how long will this take to run, let’s say, on your laptop?

(a) Milli-seconds
(b) Seconds
(c) Minutes
(d) Hours
Quizz Time!
Matrix Multiplication: Characteristics

Multiplying two matrices of size $N \times N$:

- Compute: $O(N^3)$

(a) Naive approach (pseudo-code of earlier slide)
(b) Tiled approach (much faster!)

Jia-Wei, Kung. “I/O Complexity: The Red-Blue Pebble Game.”
Block Matrix Multiplication

Matrix A \cdot \text{tile size } T \cdot \text{matrix size } N = \text{Matrix } C
for(int i = 0; i < x; i += 16)  
   A.read_block(i, k, row);  
   B.read_block(k, j, col);  
   for (int t=0; t < 16; t++)  
      sum += row[t] * col[t]
Use High-Level Synthesis (HLS)!!!

```
for(int i =0; i < x; i += 16)
    A.read_block(i, k, row);
for(int t=0; t < 16; t++)
    sum += row[t] * col[t]

Verilog source code
(Input to APR)
```
Benefits of High-Level Synthesis

- Enable more abstract design models
- Enable **rapid** architectural exploration
- Automatic scheduling, resource sharing, and pipelining
- Usage of C++ testbench to verify HLS-generated RTL
- Easier retargeting to different silicon technologies
- Enable more debugging to occur at higher abstraction level
### A few prominent HLS tools

<table>
<thead>
<tr>
<th>HLS Tool</th>
<th>Owner</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catapult</td>
<td>Mentor Graphics</td>
<td>Commercial</td>
</tr>
<tr>
<td>Stratus</td>
<td>Cadence</td>
<td>Commercial</td>
</tr>
<tr>
<td>Vivado HLS</td>
<td>Xilinx</td>
<td>Commercial</td>
</tr>
<tr>
<td>Intel HLS</td>
<td>Intel</td>
<td>Commercial</td>
</tr>
<tr>
<td>Synphony HLS</td>
<td>Synopsys</td>
<td>Commercial</td>
</tr>
<tr>
<td>LegUp</td>
<td>University of Toronto</td>
<td>Academic</td>
</tr>
<tr>
<td>Bambu</td>
<td>Politecnico di Milano</td>
<td>Academic</td>
</tr>
<tr>
<td>GAUT</td>
<td>Université Bretagne Sud</td>
<td>Academic</td>
</tr>
</tbody>
</table>
Example of HLS-based Chip Design

ML frameworks

- Adaptive Floating-Point Quantization
- Weight Clustering

Inputs

Pretrained model

Ground Truth

William Shakespeare was an English poet, widely regarded as the greatest writer in the English language.

HLS environment

- C++ testbench
- #include <nvhls_verify.h>
- SC_MODULE(ASR_Accel) {
- Testbench(sc_module name name) {
- ...
- }
- }

- C++
- testbench

- RTL verification correct?
- RTL PPA satisfactory?

C++

- #include <nvhls_verify.h>
- SC_MODULE(ASR_Accel) {
- Testbench(sc_module name name) {
- ...
- }
- }

- C++
- testbench

Same?

No

Yes

HLS

APR

FlexASR SystemC Implementation

MatchLib

HLSLibs

Inputs

Pretrained model

Ground Truth

William Shakespeare was an English poet, widely regarded as the greatest writer in the English language.
Chip Design Flow

Input source code description of hardware design (Verilog)

Also called automatic place and route (APR)
Chip Design Flow

Also called automatic place and route (APR)

Input source code description of hardware design (Verilog) 
Generated via HLS!

Chip Layout
Synthesis

Generates gate-level netlist from Verilog language
Floor Planning

Generates gate-level netlist from Verilog language

- Aspect ratio definition of design
- Placement of memories
- Routing of power supplies

example
Placement

- Generates gate-level netlist from Verilog language
- Aspect ratio definition of design
- Placement of memories
- Routing of power supplies
- Placement and placement optimization of standard cells
Clock Tree Synthesis

Generates gate-level netlist from Verilog language

Placement and placement optimization of standard cells

• Aspect ratio definition of design
• Placement of memories
• Routing of power supplies

• Clocking optimization
• Timing optimization
• Clock gating
Routing

Generates gate-level netlist from Verilog language

- Aspect ratio definition of design
- Placement of memories
- Routing of power supplies

Placement and placement optimization of standard cells

- Clocking optimization
- Timing optimization
- Clock gating

Timing and congestion-driven detail routing
Signoff

Generates gate-level netlist from Verilog language

- Aspect ratio definition of design
- Placement of memories
- Routing of power supplies

Placement and placement optimization of standard cells

- Clocking optimization
- Timing optimization
- Clock gating

Timing and congestion-driven detail routing

- Timing optimization with post-layout extraction
- Fixing of design rules
Place-and-route is automated (Easy)

This flow can be automated!
Remember it is called **automatic** place-and-route
Use HLS for fast accelerator design!!!

Generating the input source code requires the most effort!

Use HLS for fast prototype!!

Input source code description of hardware design (Verilog)

This flow can be automated!
Remember it is called **automatic** place-and-route
Say we need to quickly design a computationally-efficient AI processor. What should we do to produce the Verilog source code of the processor?

(a) Do your best to hand-write the Verilog source code manually
(b) First, design the source code in C/C++ and then use High-Level Synthesis (HLS)
(c) Design the matrix multiplication datapath in a block or tile fashion
(d) Do both (b) and (c)
Resources

- High Level Synthesis Blue Book: [https://www.cse.usf.edu/~haozheng/teach/cda4253/doc/hls/hls_bluebook_uv.pdf](https://www.cse.usf.edu/~haozheng/teach/cda4253/doc/hls/hls_bluebook_uv.pdf)

- Example of HLS-based domain-specific accelerator (FlexASR): [https://github.com/harvard-acc/FlexASR](https://github.com/harvard-acc/FlexASR)

- Physical Design via Place-and-Route: [https://inst.eecs.berkeley.edu/~ee290c/sp18/lec/Lecture12A.pdf](https://inst.eecs.berkeley.edu/~ee290c/sp18/lec/Lecture12A.pdf)
Thank You!