AISC
Approximate Instruction Set Computer

Alexandra Ferrerón, Darío Suárez-Gracia
Jesús Alastruey-Benedé, Ulya R. Karpuzcu

University of Zaragoza
University of Minnesota, Twin-Cities

WAX’18, March 25 2018
Instruction Set Architecture (ISA)

- View from hardware stack:
  - Behavioral design specification
  - **Determines hardware complexity**
- View from software stack:
  - Definition of the machine capabilities
  - **Determines functional completeness**
Approximate Instruction Set Computer (AISC)

• Single-ISA heterogeneous fabric
• Each compute engine (core or fixed function) supports a subset of the ISA
  • Component (functionally-incomplete) ISA may overlap
  • Incomplete ISA can reduce microarchitectural complexity
    • thereby improve performance per Watt
• The union of incomplete ISA subsets renders a functionally complete ISA
Approximate Instruction Set Computer (AISC)

- Single-ISA heterogeneous fabric
- Each compute engine (core or fixed function) supports a subset of the ISA
  - Component (functionally-incomplete) ISA may overlap
  - Incomplete ISA can reduce microarchitectural complexity
    - thereby improve performance per Watt
- The union of incomplete ISA subsets renders a functionally complete ISA

Code that cannot be approximated can run at full accuracy

Improved energy efficiency if ISA-induced approximation is tolerable
Approximate Instruction Set Computer (AISC)

• How to determine incomplete, approximate ISA subsets?
  • Vertical approximation:
    • Exclude less frequently used complex instructions
  • Horizontal approximation:
    • Simplifies each instruction, by e.g., precision reduction
Proof-Of-Concept

(a) Native  (b) Vertical  (c) Horizontal  (d) Horiz.+Vert.
The key question is how to pick the instructions to drop. A good output is the Cortex suite [256]. AISC may degrade energy efficiency, but the question becomes more critical in this case. As a more general version of this question, let us start with a motivating example. Fig. 1 shows how the application is approximated under AISC. We observe that the accuracy loss remains the same for comparison.

Let us start with a motivating example. Fig. 1 shows how the Super Resolution Reconstruction, a computer vision application from the 2008 IEEE conference on Computer Vision and Pattern Recognition, is handled. The input is the (64 × 128) reconstructed image. We use the (256 × 256) reconstructed image. We use the (64 × 128) reconstructed image. We use the (256 × 256) reconstructed image.

While fast code migration is not impossible, if not orchestrated carefully, the energy overhead of AISC may degrade energy efficiency. Without loss of generality, we experimented with two representations in this case: (a) Native, (b) Vertical, (c) Horizontal, (d) Horiz.+Vert.
Proof-Of-Concept

(a) Native  (b) Vertical  (c) Horizontal  (d) Horiz.+Vert.
Proof-Of-Concept

(a) Native      (b) Vertical      (c) Horizontal      (d) Horiz.+Vert.

double precision $\Rightarrow$ half-precision
Without loss of generality, we experimented with three approaches with threshold values between 1% and 10%.

The key question is how to pick the instructions to drop. A (graphic) output of a typical noise-tolerant application, Super Resolution Reconstruction, a computer vision application from the WAX'18, March 25, 2018, conference, is shown.

Let us start with a motivating example. Fig. 1 shows how the following discussion.

Figure 1. REPRESENTATIVE AISC APPROXIMATIONS

(a) Native (b) Vertical (c) Horizontal (d) Horiz. + Vert.

DIV \rightarrow MUL

DIV to MUL, DIV to MUL.NR, DIV to MUL12, MUL to ADD, MUL to ADD.NR

DIV to MUL converts division instructions to multiplication instructions. MUL to ADD converts multiplication instructions to a sequence of additions. We chose the smaller of the factors as the multiplier (which determines the number of additions), and the reciprocal (of only 12 bit precision). Fig. 1(d) captures an example execution output of a typical noise-tolerant application, SRR, with a break-even point on Pin 2.14 for comparison,

On the other hand, reliance on one iteration of the Newton-Raphson procedure for our proof-of-concept and leave for future work.

Our proof-of-concept analysis revealed that, in its restricted form – where the region of interest of an application is barely visible, but still varies across different approximations.

Therefore, conservatively assume that this entire code would be approximated carefully, the energy overhead of

- DP SPD to HP
- DP to SP
- SP to HP
- SP to DP

can easily become prohibitive. Therefore, a break-even point

Between the course of its execution – AISC can cut energy up to 37% at around 10% accuracy loss.

Approximate under AISC

WAX'18, March 25, 2018

University of Minnesota, Twin Cities

Alexandra Ferrerón

Universidad de Zaragoza, Spain


The key question is how to pick the instructions to drop. A
output is the (256 Cortex suite [2.2 point instructions. For each static instruction, we based the
struction dropping to data-phases as opposed to control [the most aggressive in our bag of tricks,
sions, but the question becomes more critical in this case. As
more general version of this question,
2.1 representative AISC approximations (b)-(d).

Let us next take a closer look at the sources of this diversity.
barely visible, but still varies across di
line for comparison,
errors or simply enforced by design. The latter applies for
WAX'18, March 25, 2018

Super Resolution Reconstruction, a computer vision application from the
Proof-Of-Concept

(a) Native  (b) Vertical  (c) Horizontal  (d) Horiz.+Vert.

Up to 37% energy cut at around 10% accuracy loss
Design Aspects

• Which subset of the ISA should each compute engine support?
• How to map instruction sequences to compute engines?
• How to keep the potential accuracy loss bounded?
• How to orchestrate migration of code sequences
  • from one compute engine to another within the course of computation
  • tolerance to noise may vary for different application phases
Design Aspects

• Which subset of the ISA should each compute engine support?
• How to map instruction sequences to compute engines?
• How to keep the potential accuracy loss bounded?
• How to orchestrate migration of code sequences
  • from one compute engine to another within the course of computation
  • tolerance to noise may vary for different application phases

• Most critical design aspect: migration granularity
  • A break-even point exists for migration granularity (and frequency)
AISC
Approximate Instruction Set Computer

Alexandra Ferrerón, Darío Suárez-Gracia
Jesús Alaстрuey-Benedé, Ulya R. Karpuzcu

ukarpuzc@umn.edu

WAX’18, March 25 2018
## Instruction Set Architecture (ISA)

### Software

```c
int main(void)
{
    ...
}
```

### Architecture

- **DIV.D**: F0, F2, F4
- **ADD.D**: F10, F0, F8
- **SUB.D**: F8, F8, F14

### Hardware

- [Fetch]
- [Decode]

---

**Note:**
- The diagram illustrates the fetch and decode stages of the instruction execution process.
- The instructions `DIV.D`, `ADD.D`, and `SUB.D` are examples of operations that can be performed in the architecture.