AISC: Approximate Instruction Set Computer

Alexandra Ferrerón¹, Jesús Alastruey-Benedé¹, Darío Suárez-Gracia¹, Ulya R. Karpuzcu²

¹ Universidad de Zaragoza, Spain ² University of Minnesota, Twin Cities

Abstract
This paper makes the case for a single-ISA heterogeneous computing platform, AISC, where each compute engine (be it a core or an accelerator) supports a different subset of the very same ISA. An ISA subset may not be functionally complete, but the union of the (per compute engine) subsets renders a functionally complete, platform-wide single ISA. Tailoring the microarchitecture of each compute engine to the subset of the ISA that it supports can easily reduce hardware complexity. At the same time, the energy efficiency of computing can improve by exploiting algorithmic noise tolerance: by mapping code sequences that can tolerate (any potential inaccuracy induced by) the incomplete ISA-subsets to the corresponding compute engines.

1 Motivation
The ISA specifies semantic and syntactic characteristics of a practically functionally complete set of machine instructions. Modern ISAs are not necessarily mathematically functionally complete, but provide sufficient expressiveness for practical algorithms. For software layers, the ISA defines the underlying machine – as capable as the variety of algorithmic tasks the composition of its building blocks, instructions, can express. For hardware layers, the ISA rather acts as a behavioral design specification for the machine organization. Accordingly, the ISA governs both the functional completeness and complexity of a machine design.

This paper makes the case for an alternative, single-ISA heterogeneous computing platform, AISC, which can reduce the ISA complexity, and thereby improve energy efficiency, on a per compute engine (be it a core or an accelerator) basis, without compromising the functional completeness of the overall platform. The distinctive feature of AISC is that each compute engine supports a different subset of the very same instruction set. Such per compute engine ISA subsets may be disjoint or overlapping. An ISA subset may not be functionally complete, but the union of the (per compute engine) subsets renders platform-wide a functionally complete single ISA. Therefore, software layers perceive AISC as a single-ISA machine. Tailoring the microarchitecture of each compute engine to the subset of the ISA that it supports results in less complex, more energy efficient compute engines, without compromising the overall functional completeness of the machine.

When it comes to the design of a feasible AISC platform, many questions arise, the most critical being:

- Which subset of the ISA should each compute engine support, by construction?
- How to guarantee that each sequence of instructions scheduled to execute on a given compute engine only spans the respective subset of the ISA (with potential accuracy loss)? More specifically, how to map instruction sequences to the compute engines?
- How to orchestrate migration of code sequences from one compute engine to another within the course of computation, as different application phases may exhibit different degrees of tolerance to noise?
- How to keep the potential accuracy loss bounded?

Starting with the first and most basic question, approximation along two dimensions can set the ISA subset per compute engine:

- Horizontal approximation simplifies instructions by reducing complexity (e.g., precision) on a per instruction basis. To be more specific, the subset of the ISA a compute engine implements in this case would selectively contain lower complexity (e.g., lower precision) instructions, by construction. Well-studied precision reduction approaches [2, 5, 7, 9–11, 13, 15] are directly applicable in this context. Reducing the operand width often enables simplification in the corresponding arithmetic operation, in addition to a more efficient utilization of the available communication bandwidth for data (i.e., operand) transfer.
- Vertical approximation excludes complex and less frequently used instructions from the subset.
- The combination of the two dimensions, Vertical+Horizontal, is also possible: In this case, the compute engine concerned would be able to approximately emulate complex and less frequently used instructions (that its ISA subset does not contain) by a sequence of simpler instructions.

AISC trades computation accuracy for the complexity (and thereby, energy efficiency) on a per compute engine basis. At the same time, as the entire platform still supports the full-fledged ISA, instruction sequences not prone to approximation can still run at full accuracy. AISC can also be regarded as an aggressive variant of architectural core salvaging [8] or ultra-reduced instruction set coprocessors [14], where actual hardware faults impair a compute engine’s capability to implement a subset of its ISA (and all compute engines support the same ISA by design). These lines of studies detail how to achieve full-fledged functional completeness under the hardware-fault-induced loss of support for a subset of instructions. AISC, on the other hand, features compute engines with approximate, i.e., incomplete or restricted, ISAs by construction. Without loss of generality, such incomplete or restricted ISAs within an AISC platform may be due to...

WAX’18, March 25, 2018
Workshop on Approximate Computing at ASPLOS 2018.
errors or simply enforced by design. The latter applies for
the following discussion.

2 Proof-of-concept Implementation

Let us start with a motivating example. Fig. 1 shows how the
(graphic) output of a typical noise-tolerant application, SRR 1,
changes for representative Vertical, Horizontal, and Horizon-
tal + Vertical approximation under AISC. The application is
compiled with GCC 4.8.4 with -O1 on an Intel® Core™ i5
3210M machine. As we perform manual transformations on
the code, high optimization levels hinder the task; we resort
to -O1 for our proof-of-concept and leave for future work
more exploration on compiler optimizations. We focus on
the main kernel where the actual computation takes place, and
conservatively assume that this entire code would be
mapped to a compute engine with approximated ISA. We
use ACCURAX metrics [1] to quantify the accuracy-loss. We
prototype basic Horizontal and Vertical ISA approximations
on Pin 2.14 [6]. Fig. 1(a) captures the output for the base-
line for comparison. Native execution, which excludes any
approximation. We observe that the accuracy loss remains
barely visible, but still varies across different approximations.
Let us next take a closer look at the sources of this diversity.

Figure 1. Graphic output of SRR benchmark under repre-
sentative AISC approximations (b)-(d).

2.1 Vertical Approximation

The key question is how to pick the instructions to drop. A
more general version of this question, which instructions to
approximate under AISC, already applies across all dimen-
sions, but the question becomes more critical in this case. As
the most aggressive in our bag of tricks, Vertical can incur
significant loss in accuracy. The targeted recognition-mining-
synthesis applications can tolerate errors in data-centric
phases as opposed to control [3]. Therefore, confining in-
struction dropping to data-flow can help limit the incurred
accuracy loss. Fig. 1(b) captures an example execution out-
come, where we randomly deleted static (arithmetic) floating
point instructions. For each static instruction, we basted
the dropping decision on a pre-defined threshold $t$. We gener-
ated a random number $r$ in the range $[0, 1]$, and dropped
the static instruction if $r$ remains below $t$. We experimented
with threshold values between 1% and 10%.

2.2 Horizontal Approximation

Without loss of generality, we experimented with three ap-
proximations to reduce operand widths: DPtoSP, DP(SP)toHP,
and DP(SP)toINT. Under the IEEE 754 standard, 32 (64) bits
express a single (double) precision floating point number:
one bit specifies the sign ($\{\text{sign\, bit}\}$, the exponent; and 23 (52)
bits the mantissa, i.e., the fraction. For example, $(-1)^{\text{sign\, bit}}\times
2^{\text{exponent\, bit}} \times \text{mantissa}$ represents a single-precision float-
ing number. DPtoSP is a bit discarding variant, which omits 32
least-significant bits of the mantissa of each double-precision
operand of an instruction, and keeps the exponent intact.
DP(SP)toHP comes in two flavors. DPtoHP omits 48 least-
significant bits of the mantissa of each double-precision
operand of an instruction, and keeps the exponent intact; SP-
toHP, 16 least-significant bits of the mantissa of each single-
precision operand of an instruction. Fig. 1(c) captures an
example execution outcome under DPtoHP. DP(SP)toINT
also comes in two flavors. DPtoINT (SPtoINT) replaces dou-
ble (single) precision instructions with their integer counter-
parts, by rounding the floating point operand values to the
closest integer.

2.3 Horizontal + Vertical Approximation

Without loss of generality, we experimented with two rep-
resentatives in this case: MULtoADD and DIVtoMUL. MUL-
toADD converts multiplication instructions to a sequence of
additions. We picked the smaller of the factors as the
multiplier (which determines the number of additions), and
rounded floating point multipliers to the closest integer num-
ber. DIVtoMUL converts division instructions to multiplica-
tions. We first calculated the reciprocal of the divisor, which
next gets multiplied by the dividend to render the end result.
In our proof-of-concept implementation based on the
x86 ISA, the reciprocal instruction has 12-bit precision.
DIVtoMUL12 uses this instruction. DIVtoMULNR, on the
other hand, relies on one iteration of the Newton-Raphson
method [4] to increase the precision of the reciprocal to
23 bits. DIVtoMUL12 can be regarded as an approximate
version of DIVtoMULNR, eliminating the Newton-Raphson
iteration, and hence enforcing a less accurate estimate of
the reciprocal (of only 12 bit precision). Fig. 1(d) captures an
example execution outcome under DIVtoMULNR.

3 Conclusion & Discussion

Our proof-of-concept analysis revealed that, in its restricted
form – where the region of interest of an application is
mapped in its entirety to an incomplete-ISA compute en-
gine, irrespective of potential changes in noise tolerance
within the course of its execution – AISC can cut energy up
to 37% at around 10% accuracy loss.

The most critical design aspect is how instruction se-
quences should be mapped to restricted-ISA compute en-
gines, and how such sequences should be migrated from one
engine to another within the course of execution, to track
potential temporal changes in algorithmic noise tolerance.
While fast code migration is not impossible, if not orches-
trated carefully, the energy overhead of fine-grain migration
can easily become prohibitive. Therefore, a break-even point
in terms of migration frequency and granularity exists, past
which AISC may degrade energy efficiency.

1 Super Resolution Reconstruction, a computer vision application from the
Cortex suite [12]. We use the (64×64) “ELA” input data set of 16 frames. The
output is the (256×256) reconstructed image.
References


