surecore

WHITE PAPER 4

Improving the Efficiency of AI Applications Using In-Memory Computation

TONY STANSFIELD
CTO, SURECORE LIMITED

enquiries@sure-core.com sureCore Ltd, Sheffield Technology Park, Cooper Buildings, Arundel Street, Sheffield S1 2NS, UNITED KINGDOM
www.sure-core.com

Improving The Efficiency Of AI Applications Using In-Memory Computation

Tony Stansfield, CTO, SureCore Limited

This white paper introduces sureCore's *CompuRAM*™ platform – SRAM architecture extensions to support In-Memory Computation. This is a useful technology for improving the power-performance of AI applications, and is particularly applicable to power-constrained applications – for instance when AI techniques are used to improve the functionality of stand-alone devices or battery-powered but network-connected devices. The white paper explains why SRAM and In-Memory Computation are important for AI applications, then explains some issues relevant to SRAM circuit optimisation for AI, before moving on to the properties of Artificial Neural Networks and the vector dot product, and the features of *CompuRAM* that support them. A definition of In-Memory Computation and how it differs from other ways to implement high-performance computation is also included.

Artificial intelligence and the need for efficient SRAM

In recent years, Artificial Intelligence (AI) has become a significant user of SRAM, as can be seen in the development of large, massively parallel, server chips that contain hundreds or thousands of individual processors. Each processor has its own working memory and the overall chip can contain 10s or 100s of Mbytes of SRAM, leading to SRAM being the single largest component of chip area in these devices. This means that even minor improvements in SRAM area, power, or reliability can have a noticeable impact on chip-level performance.

AI is increasingly moving out of the datacentre and into edge devices. For example:

- Adaptive control of noise cancellation in earbuds, to suppress background noise while still allowing through important sounds (sirens, babies crying, etc).
- Pre-processing of images in security cameras to identify relevant changes (e.g. people entering or leaving the field of view) and to screen out false positives (such as pets walking across a lawn)

These applications are typically adding additional functionality to already power-constrained products and the ideal scenario is to add these new features without affecting battery life. Therefore, SoCs developed for these applications have different requirements to the datacentre AI chips – peak compute capability is no longer the key goal, with peak power and energy per operation being more relevant. Such SoCs have different architectures that trade flexibility for efficiency. For instance, an increasingly common goal in edge AI is to exploit some form of 'In-Memory Compute' (IMC). This entails moving some computational resources away from the processor and into the memory system (see 'What is In-Memory

Computation?' for more details). This increases power efficiency by reducing bus traffic but requires the correct placement of data in memory in order to be able to fully use those computational resources.

An SRAM developer's view of memory for AI

sureCore is an established supplier of low-power SRAM and register file IP and related design services. We provide both standard products and optimised memory subsystems for specific applications. AI, and especially edge AI, is a potentially large market with some specific properties that make it worth re-optimising the memory subsystem to support it (see Table 1 for some examples of possible memory optimisations). sureCore develops application-optimised subsystems in partnership with customers who understand the application space – a combination of their application knowledge and sureCore's deep memory expertise produces a better result than either could achieve in isolation. From a sureCore perspective, such partnerships use our existing technologies as a platform to build on, with application knowledge guiding the choice of which technologies to use, and how to best combine them to meet the overall system goals. sureCore has also been investigating how to extend this technology platform to support other features useful for edge AI – in particular, what we can do to support In-Memory Computing. The rest of this white paper describes these features, starting firstly with a description of the properties of AI applications and then moving on to the implications for SRAM and for IMC.

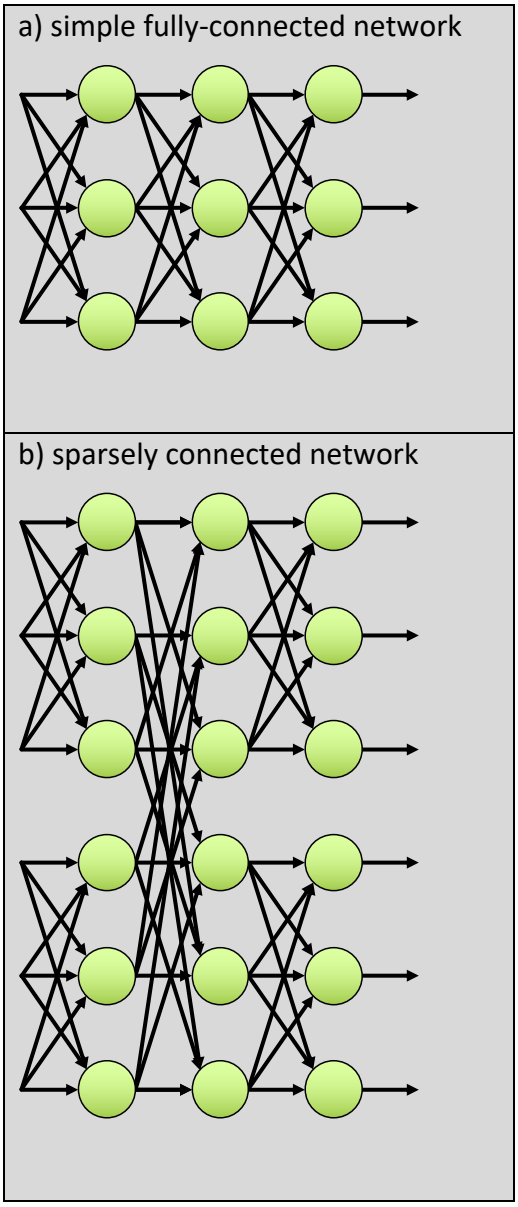
Table 1 Possible SRAM tradeoffs related to AI applications

Trade-off	***Description***	***Usefulness to AI***
Read power vs Write power	Circuit optimisations can reduce read power at the expense of increased write power (or vice versa). Usefulness of these optimisations depends on ratio of reads to writes	In deployed applications weights are read frequently, but only rarely updated, therefore have very skewed read/write ratio, so read power optimisation is attractive
Leakage vs active power	It is possible to reduce leakage at the expense of active power, or at the expense of time to switch in and out of sleep states	Depends on proportion of time the application is asleep rather than active. Also depends on frequency of switching states
Speed vs bandwidth vs area	Small (low capacity) memory instances are faster than larger ones, but larger ones are more area efficient for the same total storage. Also, a large memory with a wide word has a higher total bandwidth than a narrower and faster memory.	AI has deterministic memory access patterns, so aggregate bandwidth is likely more important than speed for a single access

AI algorithm architecture

Artificial Neural Networks are the most common architecture used for AI applications. Such networks consist of a large number of interconnected 'neurons,' where each neuron has an output and one or more inputs. The output value is generated from a weighted sum of input values. Figure 1 shows some example neural networks with neurons represented as green circles and arrows representing the input and output connections.[1] Figure 1a is a very simplified example where neurons are arranged in layers (columns) and in each layer every neuron is connected to all the inputs to that layer. However, this structure is not used in practice as it does not scale well – the total number of weights that need to be set (and to be stored) per layer is equal to the number of inputs multiplied by the number of neurons. More practical networks are sparsely connected so that each neuron only connects to a subset of the possible inputs – more like Figure 1b. There are some useful features of such sparsely connected networks:

Figure 1 Example neural networks



- There are clusters of neurons that share the same inputs
- Not all layers follow the same clustering pattern – there are layers that take inputs from across the clusters in a previous layer
- One common case is the 'Convolutional Neural Network', where equivalent neurons in different clusters use the same weights. This significantly reduces the number of different weights that have to be chosen and to be stored.

Convolutional neural networks are potentially much simpler to support with dedicated hardware, as the reuse of weights across neurons mean that the same hardware can be used to process multiple neurons – increasing the size of the input does not automatically require more weight memory, or more computing hardware, as the larger input can use the same hardware resources but take longer to process.

[1] Note that these diagrams are for illustration only. The precise network topology in an ANN is outside the scope of this white paper, except where it has an impact on the requirements for hardware used to implement the network.

A key algorithm for AI and its implications for IMC architecture

A key algorithm for AI applications is the vector dot product[2] – pairwise multiplication of two lists of numbers, followed by summing the results. This is a building block of matrix multiplication, which in turn is a building block for describing artificial neural networks. An efficient implementation of this algorithm is therefore an important component of an IMC platform.

The dot product[3] of two vectors ***a*** and ***b*** can be defined as:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{k=0}^{n-1} a_k b_k = a_0 b_0 + a_1 b_1 + \cdots + a_{n-1} b_{n-1}$$

Note that each of the individual numbers a_k (or b_k) is itself a binary-weighted vector of bits:

$$a_k = \sum_{i=0}^{l-1} 2^i \, a_{k,i} \quad \text{and} \quad b_k = \sum_{j=0}^{m-1} 2^j b_{k,j}$$

So:

$$a_k b_k = \left(\sum_{i=0}^{l-1} 2^i a_{k,i} \right) \left(\sum_{j=0}^{m-1} 2^j b_{k,j} \right) = \sum_{i=0}^{l-1} \sum_{j=0}^{m-1} 2^i 2^j a_{k,i} b_{k,j}$$

And therefore:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{k=0}^{n-1} a_k b_k = \sum_{k=0}^{n-1} \sum_{i=0}^{l-1} \sum_{j=0}^{m-1} 2^i 2^j a_{k,i} b_{k,j} = \sum_{i=0}^{l-1} 2^i \sum_{j=0}^{m-1} 2^j \sum_{k=0}^{n-1} a_{k,i} b_{k,j}$$

The last step changes the summation order, and can be read from right-to-left as:

- Form the individual bit products across a_k and b_k.
- Sum equivalent bit products from different words.
- Then shift and add these partial results to produce the final result.
    - The shift and add can be a 2-stage process
        - Sum over the bits of b (the stored coefficients)
        - Then sum over the bits of the input data stream.

If done in this way, then the sum over the input data stream can itself be implemented serially – separate bits of the a_k inputs can be sent in separate cycles. This simplifies hardware requirements, at the expense of increased execution time.[4]

---

[2] Vector dot product is the same operation as Multiply-Accumulate (MAC), or the SUMPRODUCT() function in a spreadsheet.

[3] In AI, one of the input vectors (e.g. ***a***) is variable, coming directly from the input data or being derived from it. The other vector (***b***) is a set of fixed coefficients (usually referred to as *weights* in the AI context).

[4] Serialising data streams in this way is a known way of trading hardware complexity for run time. It is especially useful in multiplication-intensive applications where the size of a fully parallel hardware multiplier scales with the product of the word lengths of the two inputs and can be wasteful if required to support more than one word size. Serialising one or more inputs reduces the hardware cost but requires more cycles per multiplication (but can allow the use of faster clock rates to mitigate some of this cost).

Introducing *CompuRAM* – the sureCore IMC platform

The list of features at the end of the previous section forms the basis of *CompuRAM*, which:

1. Supports bitwise logical operations to combine stored and input data
2. Efficiently forms partial sums of the results of these bitwise operations
3. Supports shift-and-add operations on these partial sums
4. Uses serial data transfer to move data in and out of IMC-capable memories.

Items 1 and 2 in this list are implemented with efficient hardware embedded in the bit cell arrays of a modified version of sureCore's existing *PowerMiser*™ low-power SRAM architecture. Shift and add operations can be implemented in the SRAM periphery using custom logic that is tightly integrated with the SRAM I/O circuits. Finally, serial data transfer can be implemented using a combination of synthesised logic and sureCore's multiport register file technology.

PowerMiser™ saves power in two basic ways:

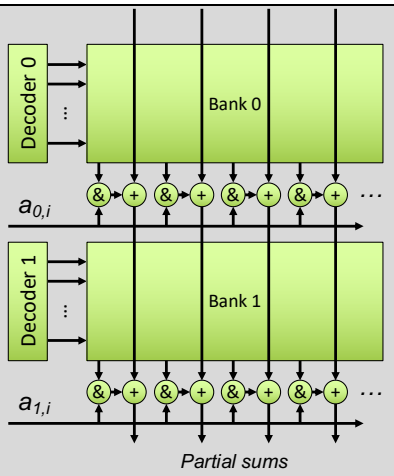
1. It is a modular architecture and ensures that only those components that are needed for each memory access are active.
2. It uses energy-optimised signalling techniques to move data between the modules.

Figure 2 shows how a *PowerMiser*™ core is modified to implement IMC operations. Firstly, multiple modules (e.g., the Banks in Figure 2) can be activated at once to get higher-bandwidth access to the coefficient store. Then the necessary extra circuits to combine coefficients and external inputs are added in, but the power-efficient signalling is retained for moving data within the memory.

Platform flexibility is provided by:

- Choice of the sum limits *l*, *m*, *n* in the preceding equations (respectively the width of an input word, the width of a stored coefficient, and the length of the vectors ***a***, ***b***). These can all be set according to the needs of an individual application or left with some flexibility in order to allow an SoC to be adapted in the future.
- Choice of memory sizes – i.e., the number of coefficients that can be stored
- Choice of how data flows between IMC memories – i.e., the form of the synthesised logic referred to above.

Figure 2 PowerMiser™ core with IMC additions



Banks in the memory store multiple sets of coefficients. The decoders are used to access the coefficients needed for a particular calculation, which are then combined with bits of the ***a*** input words as shown.

A single bit from ***a*** is combined with multiple coefficient bits. These can be multiple bits of the same word, b_k, or coefficients from multiple artificial neurons that share the same inputs.

What is In-Memory Computation?

Figure 3 illustrates four possible approaches to combining processors, memory, and logic in an SoC in order to achieve high computational performance.

These four approaches can be distinguished by the answers to the following questions:

1. How much simultaneous processing is there?

   Figure 3(a) uses a single processor and is therefore limited by single-processor performance and by the ability to move data to/from that processor fast enough. The other examples rely on parallel operation and therefore require that there is sufficient intrinsic parallelism in the application for this to be possible

2. Which side of the memory bus does the processing take place?

   In Figure 3(c) and (d), some processing logic is moved to the memory side of the bus rather than the processor side. This has two principal advantages:

   a. Traffic on the bus is reduced. For instance, in the case of the vector dot product example, it is only the result of the dot product that needs to be sent to a processor, not all the individual components of each vector. Reduced bus traffic saves power, and/or frees up bus capacity for other purposes.
   b. The processor word width no longer limits the amount of data that is processed in each cycle. Instead, a wider word based on the properties of the memory system can be used.[5] This increases the use of available parallelism.

   However, there are also some disadvantages:

   a. Using a different word width to that used by the processors creates potential issues with ensuring that data is correctly aligned.
   b. ALU(s) close to the memory are good for data processing, but not for control-intensive processing (i.e., running code with lots of if…then…else conditions).

3. Does each ALU have the same (unified) memory view?

   In Figure 3(a), (b) and (c), all ALUs and processors connect to the same memory bus and have equivalent ease of access to all addresses. However, in Figure 3(d) ALUs are closely coupled to subsections of the memory and have limited or no access to other subsections. While this arrangement provides higher bandwidth between ALU and memory than any of the others, it requires that data is correctly distributed across the different subsections in order to actually exploit this available bandwidth.

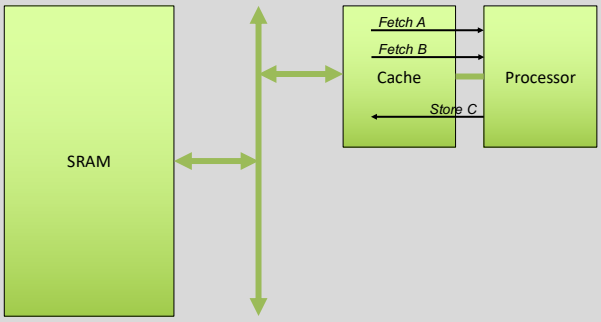
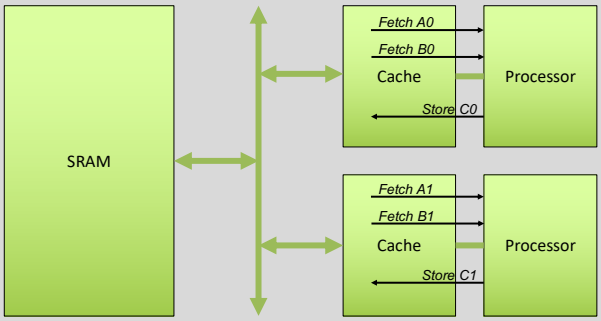
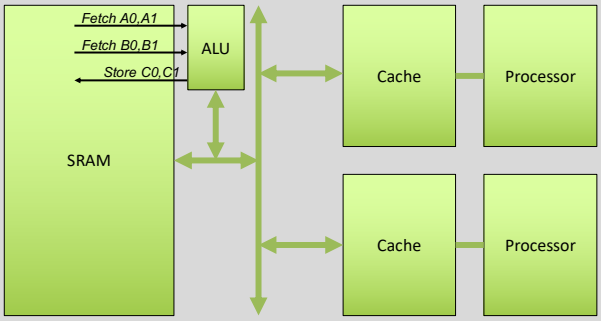
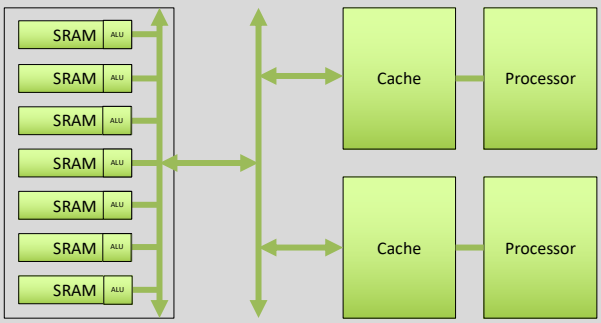
Taken together, these questions show that In-Memory Computation is only effective if an application:

- Has significant amounts of data parallelism and does not require a lot of control code
- Has predictable memory access patterns, allowing data to be correctly distributed across multiple memories

[5] In Figure 3(c) the ALU can operate on complete cache lines, while in Figure 3(d) even wider data words are possible

AI is an application space that relies heavily on matrix and vector operations hence making it highly amenable for to the benefits provided by In-Memory Computation.

Figure 3 Four possible SoC compute architectures

	(a) Single processor architecture. • Processor operates on single words coming from register file, cache, or other on-chip memory • Performance is ultimately limited by memory bus bandwidth for large data sets (i.e., that don't fit in the cache)
	(b) Shared memory parallel processor • Separate processors work on different parts of the data set • Performance still limited by memory bus bandwidth for large data sets • Requires processing to be organised so that different parts of the data can be in different caches
	(c) 'Near-memory computation' • ALU functionality added on the memory side of the bus • Good for data processing, not for control-intensive code • Uses word width of memory system (cache lines), not CPU • Processor(s) and memory bus are free for other tasks
	(d) 'In-memory computation' • ALUs closely coupled with individual memory banks in the on-chip SRAM • Increases available bandwidth between storage and ALUs – not limited by processor or cache • Requires that data is distributed across the multiple banks in order to fully exploit available ALUs

Tony Stansfield

Tony has over 35 years of semiconductor industry experience in a variety of technical roles. He started his career with the Inmos UK Memory and Graphics group, where he designed SRAMs and Caches for multiple Inmos products. He later joined HP Labs to work on high-speed programmable imaging datapaths, and was a co-founder and VP Hardware Architecture at Elixent, the company created to deliver custom Silicon IP based on that technology. Following the acquisition of Elixent by Panasonic, he was a key member of the team that integrated this technology into multiple generations of TV chipsets. Tony is cited as an inventor on 23 patents covering SRAM, CAM, low-power electronics, and programmable logic.

sureCore™ -- *When low power is paramount*™

sureCore, the ultra-low power, embedded memory specialist, is the low-power innovator who empowers the IC design community to meet aggressive power budgets through a portfolio of ultra-low power memory design services and standard IP products. sureCore's low-power engineering methodologies and design flows meet the most exacting memory requirements with a comprehensive product and design services portfolio that create clear market differentiation for customers. The company's low-power product line encompasses a range of close to near-threshold, silicon proven, process-independent SRAM IP.

www.sure-core.com