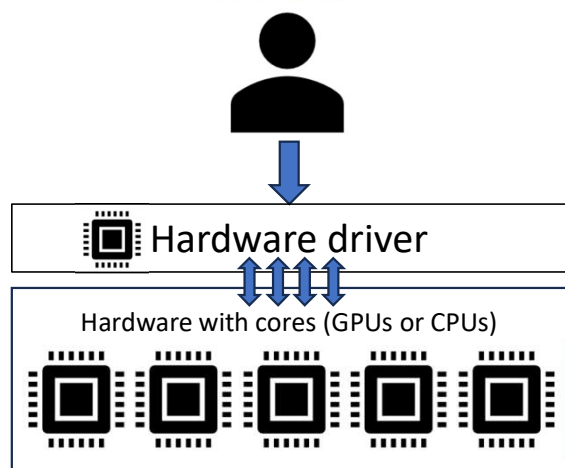# paceval.®

Faster mathematics with even lower power consumption, implemented on a **mathematical engine in hardware**.
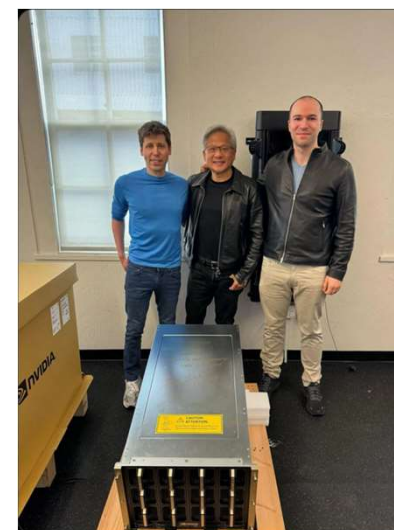
Vision Paper
New efficient coprocessor in hardware for local intelligence
2024

Today's standard approach

Nowadays, hardware with a large number of processor cores is used to calculate inference in artificial intelligence. GPUs such as the H200 from Nvidia have **several thousand processor cores** ▣. Special solutions such as those from Cerebras **even have several hundred thousand** processor cores.

Nvidia DGX H200 delivered to OpenAI by Nvidia CEO
(Picture: Greg Brockman/X)

The size of the systems and their **power consumption is very high** for operating the processor cores and managing them. **Scaling the entire system is therefore difficult**.

paceval.®

"The hardest thing is: **how does that (technology) fit in to a cohesive, larger vision**, that's going to allow you to sell 8 billion dollars, 10 billion dollars of product a year?

And, one of the things I've always found is that **you've got to start with the customer experience** and work backwards for the technology."

Steve Jobs, 1997

What is *paceval.*?

paceval. **is a mathematical engine** that can calculate almost any complex mathematical expressions. The software reads a textual description of the expressions as a mathematical function, which may contain the basic arithmetic operations, the usual transcendental functions (trigonometry, exponential function, etc.) and other common operations.

Expressions can contain **any number of placeholders (variables)**. Calculations are performed in selectable precision (32bit, 64bit and 80bit) and **distributed across all available processors** for maximum speed and effectiveness.

Additionally, *paceval.* can also output an interval indicating the error limits due to the limited precision of floating-point number formats.

paceval.®

How does paceval work internally?

paceval. internally **creates and processes linked lists of atomic calculations** that represent the user's mathematical expressions. Creating and processing an expression as a linked list offers many advantages over the usual approach*, especially speed. The processing of the linked list when actually performing a calculation with values for the variables is done in a single C function that is called by each thread of the underlying system.

The source code performs this processing (this corresponds to the **standard cycle used by all types of processors**):

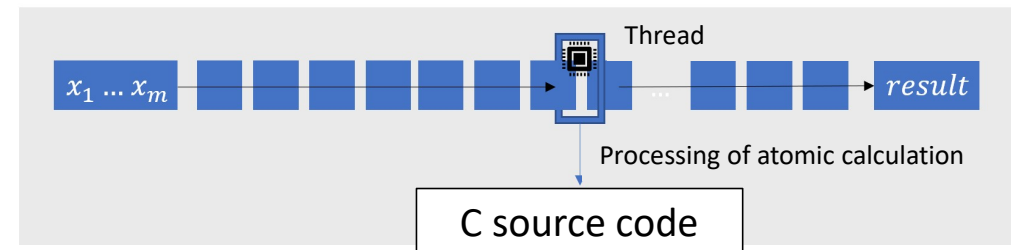**FETCH** - Get the operator and operands
(e.g. "addition of 2 and 3")

DECODE and **DECIDE** - Use the cached result from the cache or call the next step EXECUTE

EXECUTE - Execute the operator with the operands
(e.g. call the C function to add 2 and 3 and get 5 as a result)

WRITE BACK - Cache the result of the atomic calculation
(this includes lower and upper interval limits or errors)

*The usual approach is to create an expression tree from a text expression, see https://en.wikipedia.org/wiki/Binary_expression_tree. This has the well-known disadvantages such as memory consumption and the overhead and speed loss when creating and traversing the expression tree.

paceval.®

Mathematical functions

All mathematical functions can be calculated and combined with logical operators. This allows **all financial, stochastic, engineering and scientific functions and also all models for machine learning** to be represented. In addition, the usual standard mathematical notation can easily be used.

The following operators, partial functions and symbols are currently supported:
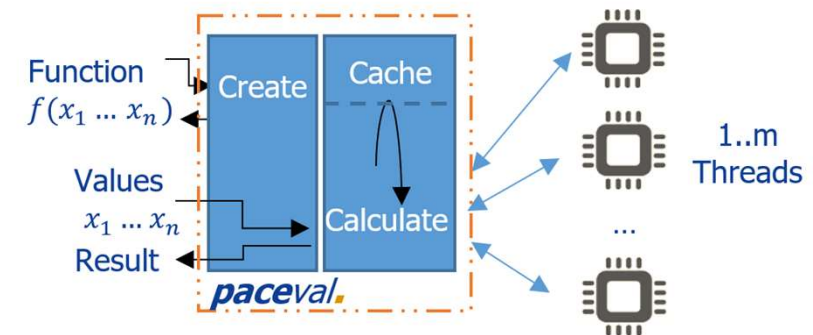- **Basic arithmetic operations** +, -, *, /
- **Logical operators** NOT, AND, OR, XOR, NAND, XNOR
- **Comparison operators** <, >, =, >=, <=, <>
- **Factorial** !, fac()
- **Constants** pi, e
- **Brackets** ()
- **Power/root functions** ^, sqr(), sqrt(), exp()
- **Logarithm functions and sigmoid function** lg(), ln(), sig()
- **Trigonometric functions and associated inverses** sin(), cos(), tan(), cot(), asin(), acos(), atan(), acot()
- **Hyperbolic functions and associated inverses** sinh(), cosh(), tanh(), coth(), arsinh(), arcosh(), artanh(), arcoth()
- **Numerical manipulations** sgn(), abs(), round(), ceil(), floor(), rand()
- **paceval specific numerical manipulations** ispos(), isposq(), isneg(), isnegq(), isnull()
- **Minimum and maximum** min, max
- **Modulo symmetric and mathematical variant** %, mod

paceval.®

"Create" and "Calculate"

Essentially **only two steps are necessary to perform calculations**. A computation object is first created ("Create" step) by the user passing a function and the set of variable identifiers. Concrete calculations can then be carried out again and again ("Calculate" step) with the object that has been created by the user passing the values for the variables.

In the **"Create" step**, a list of the individual atomic calculation rules is created with the computation object and aligned for maximum parallelizability. The user receives back a unique ID or "token" for the created object. Any number of computation objects can be created.

In the **"Calculate" step**, the calculation is carried out using the values for the variables. The list of individual atomic calculation rules is **distributed as partial sequences across all available processors or threads in the system** in order to achieve maximum speed. In addition, complex calculations of partial sequences that have been carried out once are **temporarily stored as a cache** so that they do not have to be calculated again if necessary.

Function $f(x_1 \ldots x_n)$ | Create | Cache | Calculate

Values $x_1 \ldots x_n$

Result

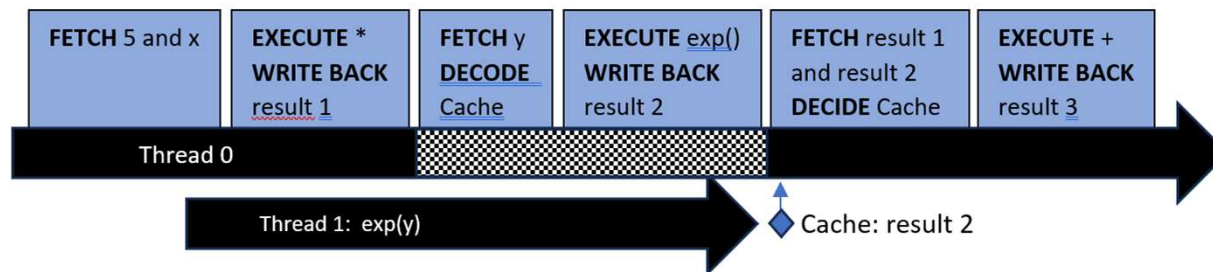*paceval.*

1..m Threads

paceval.®

A simple example in software

As an example, let's take the following function and the subsequent calculation with the given variable values:

$$f(x, y) = 5 * x + \exp(y) \quad \text{for x=2.2 und y=3.3}$$

The "Create" step will then (simplified) create this linked list in memory using the paceval library and also set **markers for possible parallelization with threads**:
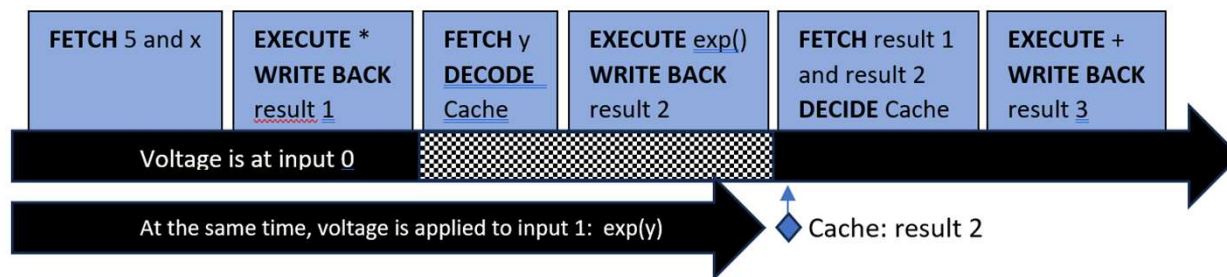

The "Calculate" step will then perform the calculation with a **total of 2 threads**. First, thread 0 is started, which has the task of performing the entire calculation, i.e. thread 0 runs through the entire linked list. Then thread 1 is started, which performs exp(y) for y=3.3 in parallel to thread 0. If thread 1 is faster, the result of the parallel calculation exp(3.3) in thread 0 is used for the final addition and **the entire calculation that thread 1 covered is skipped**.

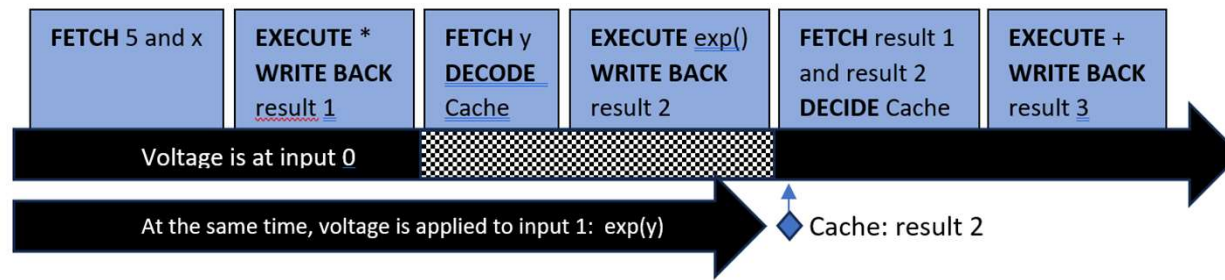**paceval®**

Now the simple example in hardware 1/2

Parallelizing the calculations in hardware has enormous advantages because **voltage is applied to the different inputs of the circuit at the same time**. To illustrate this, let's take the following function and the subsequent calculation with the specified variable values:

$$f(x,y) = 5 * x + \exp(y) \quad \text{for x=2.2 und y=3.3}$$

The "Create" step will then (simplified) create this linked list again with the paceval library as a hardware driver and provide it **with the markers for possible parallelization**. However, this list is now processed directly on the hardware, i.e. by an electronic logic circuit, analogous to the threads in the previous software implementation. The circuit will switch very quickly **and provide the result of the calculation in nanoseconds**:

paceval.®

Now the simple example in hardware 2/2

| FETCH 5 and x | EXECUTE *
WRITE BACK
result 1 | FETCH y
DECODE
Cache | EXECUTE exp()
WRITE BACK
result 2 | FETCH result 1
and result 2
DECIDE Cache | EXECUTE +
WRITE BACK
result 3 |

Voltage is at input 0

At the same time, voltage is applied to input 1: exp(y)   Cache: result 2

The "Calculate" step will then perform the calculation on **2 circuits simultaneously**. Circuit 0 has the task of performing the entire calculation, i.e. it runs through the entire list. If circuit 1 requires fewer clock cycles for exp(3.3) than 5*x, the result of the parallel calculation of exp(3.3) in circuit 1 is used for the final addition and **the entire calculation that circuit 1 covered is skipped**.

paceval.®

What hardware do we need for this approach?

This approach to parallelizing calculations in hardware was investigated in a validation assignment for SPRIND by the independent Institute for Computer Science.

For this approach, **we only need floating-point units (FPUs)** ⊞ on a hardware module, such as a standard "Field Programmable Gate Array" (FPGA) from Intel/Altera or AMD/Xilinx. Processor cores are only needed for the hardware driver. FPUs **consume far less power** than processor cores, are **inexpensive to implement**, and can be **arranged in large numbers in a small area**.
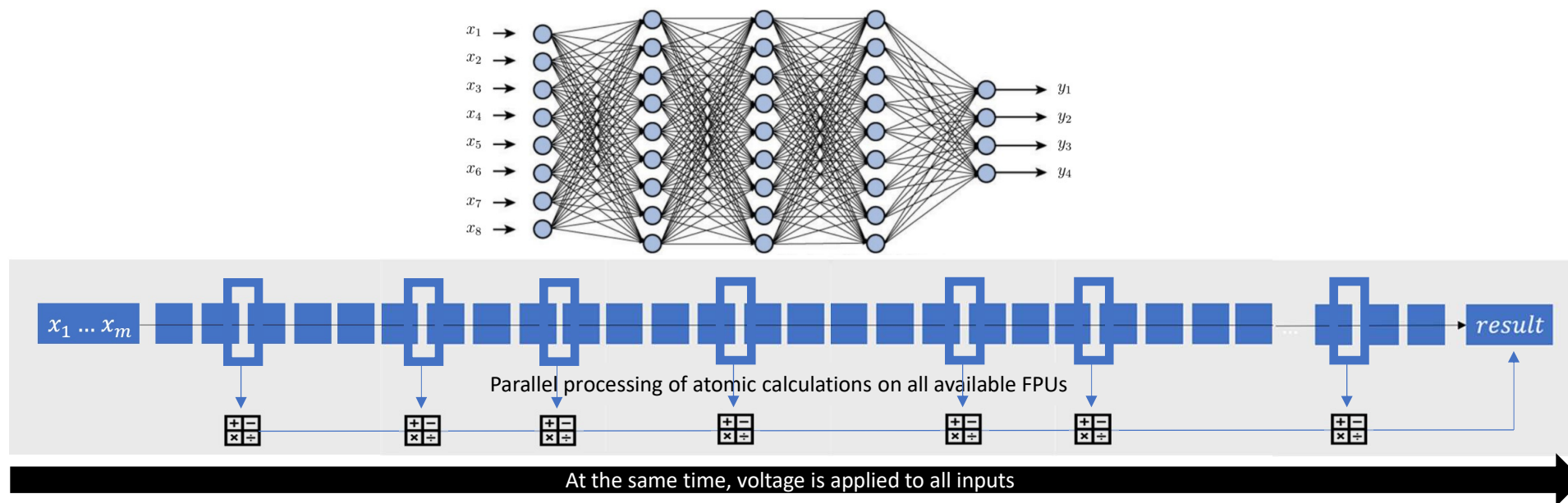
Hardware driver

Hardware with floating point units (FPUs)

new
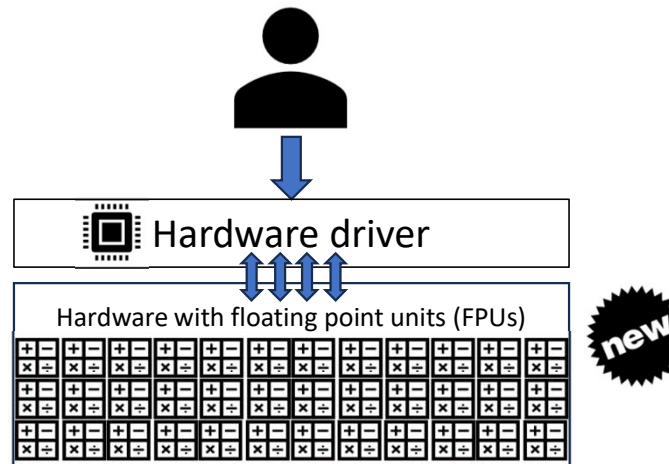
SPRIN−D
SUPPORTED BY

Bundesministerium für Bildung und Forschung

Bundesministerium für Wirtschaft und Klimaschutz

paceval®

Conclusion

Mathematical methods of artificial intelligence, especially neural networks, have great potential for parallel processing when they are presented as mathematical functions. For example, the simple neural network MNIST from example number 6 by paceval already has 372,200 possibilities for parallelization. This means that if a hardware module with 372,200 FPUs existed, **the result would be available in nanoseconds**. And all this at a **fraction of the acquisition costs, maintenance costs and power consumption** of today's standard system based on processor cores.

Parallel processing of atomic calculations on all available FPUs

At the same time, voltage is applied to all inputs

paceval.®

Power consumption in comparison

Overall, according to the research of the independent Institute of Computer Science, this **first reference design has a high energy saving potential** when executing artificial intelligence. When executing a neural network, the following comparison emerges:

TITAN X (26.7 x 11.1 cm)

	Standard 3.0 GHz CPU + NVIDIA GeForce GTX TITAN X	*paceval.* 4.24 auf CPU APPLE MAC STUDIO M1 20 Cores	*paceval.* 4.24 auf FPGA PS Zynq Ultrascale+ XCZU7EV 4 Cores	*paceval.* in Hardware/ FPGA PL Zynq Ultrascale+ XCZU7EV 96 FPUs	*paceval.* in Hardware/ASIC 256 FPUs (Values estimated)
Power consumption	400 Watt (CPU+GPU)[11]	68 Watt	3,4 Watt	3,4 Watt	< 3,4 Watt
Processing time MNIST LeNet	< 2 ms	5 ms	114 ms	< 75 ms	< 15 ms
Acquisition cost	< $ 2.000	$ 3.300	$ 850	$ 850	$ 700[12] + Profit margin
Ongoing energy costs	$ 700/Year	$ 120/Year	$5/Year	$5/Year	< $5/Year
CO_2 emission	560 kg/Year	95 kg/Year	0,9 kg/Year	0,9 kg/Year	< 0,9 kg/Year
Cooling	active	active	passive	passive	passive
Case size	> 8.000 cm³	3.700 cm³	< 50 cm³	< 50 cm³	< 35 cm³

Hardware module (5.2 x 7.6 cm)

*pace*val.®

Advantages

This new system can

- be **patented immediately**

- be developed promptly **on existing FPGA hardware**

- be **continuously improved**

- contain **customer-specific floating-point units (FPUs)** for 128bit, 64bit, 32bit, 16bit, 8bit and 1bit for scaling

This new system is

- **energy** efficient

- maximally small in **size**

- extremely **scalable**

- very **cost-effective**
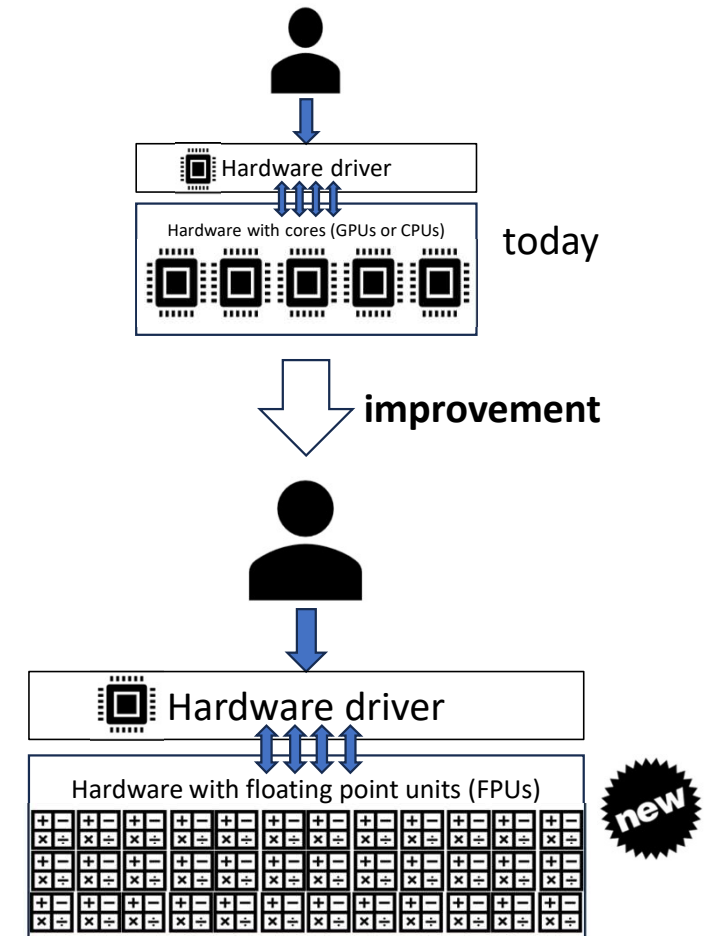
Example use case

Local intelligence in the car with the information from the manual and additional information about the specific type of car.

Hello car, which engine oil do I need and how much can I put in?

paceval®

paceval.

Create value fast.

Contact: info@paceval.com