The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations.

Level-1 BLAS perform scalar, vector and vector-vector operations

Level-2 BLAS perform matrix-vector operations

Level-3 BLAS perform matrix-matrix operations
- Optimized implementations of linear algebra kernels for several processor architectures.
- Default underlying library for many ML/DL frameworks.
- The source code distribution provides benchmarks for each BLAS kernel.
- Added **POWER10** support recently in OpenBLAS.
Most operations in training/inferencing in a neural network require some form of matrix multiplication.

Matrix Multiply Assist feature

- Eight 512 bit accumulators. Each accumulator contains four 128-bit rows.
  - 4x4 array of fp32-bit elements
  - 4x2 array of fp64-bit elements

- A set of instructions to transfer data between vector-scalar registers and accumulators.
- A set of outer product instructions that perform an outer-product operation.
Accumulators are updated by rank-k update instructions:

• Input: 1 accumulator (A) + 2 VSRs (X, Y)
• Output: 1 accumulator (same as input to reduce instruction encoding space).

**Outer-product (xv<type>ger<rank-k>) instructions**

\[
\text{ACC} = \sum (\text{opA} \times \text{opB}) + \text{ACC}
\]

- General: weight \times input
- Image: Kernel \times Pixels
The micro-kernel (innermost loop) of GEMM

\[ \mathbf{C}_{m \times n} += \mathbf{A}_{m \times k} \times \mathbf{B}_{k \times n} \]

- Load a small, “square” panel of \( \mathbf{C} \) and keep it in registers
- Load one small column of \( \mathbf{A} \) and one small row of \( \mathbf{B} \)
- Outer-product and accumulate
- Repeat!
Matrix-Multiply Assist Built-ins

- ISA 3.1 of the PowerPC added new Matrix-Multiply Assist (MMA) instructions
- GCC provides support for these instructions through the \_builtin\_mma\_* built-in functions
- Some of the builtins used now in OpenBLAS are:

\[
\begin{align*}
\text{void } \_\text{builtin}_\text{mma}_\text{xvbfs16ger2} (\text{vec}_\text{quad} *, \text{vec}_t, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xvf32ger} (\text{vec}_\text{quad} *, \text{vec}_t, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xvbfs16ger2pp} (\text{vec}_\text{quad} *, \text{vec}_t, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xfv32gerpp} (\text{vec}_\text{quad} *, \text{vec}_t, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xfv64ger} (\text{vec}_\text{quad} *, \text{vec}_\text{pair}, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xfv64gerpp} (\text{vec}_\text{quad} *, \text{vec}_\text{pair}, \text{vec}_t); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xxmtacc} (\text{vec}_\text{quad} *); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xxmfacc} (\text{vec}_\text{quad} *); \\
\text{void } \_\text{builtin}_\text{mma}_\text{xxsetacc} (\text{vec}_\text{quad} *); \\
\text{void } \_\text{builtin}_\text{mma}_\text{disassemble_acc} (\text{vec}_\text{quad} *, \text{vec}_\text{quad} *); \\
\text{void } \_\text{builtin}_\text{mma}_\text{disassemble_pair} (\text{vec}_\text{pair} *, \text{vec}_\text{pair} *);
\end{align*}
\]
Using new MMA builtins

- Hand written assembly version used in previous versions for GEMM optimization.
- Started POWER10 optimization with assembly and later converted to C code using built ins.
- Lines of code reduced from 6K to 1K for inner gemm kernels.
- Performance is closer to assembly version.
- These builtins are also now used in Eigen.

* Up to 4x improvements noted in simulator depending on various factors compared to previous processor.
Thank You!

References:

- https://github.com/xianyi/OpenBLAS/tree/develop/kernel/power