Optimizing Matrix Math: Batch-Reduced GEMM (BRGEMM) for Accelerated Deep Learning on Arm HPC Systems

Shreyas K S, Deeksha K, Vineel A G, Abhishek J, Kentaro K, Masahiro D, Priyanka S

Fujitsu Limited

Abstract

Matrix multiplications serve as a basic building block for models like Transformers and large language models (LLMs), thus contributing majorly for the performance of deep learning workloads. Among the matrix multiplication algorithms, BRGEMM (Batch-Reduced GEMM) stands out as a highly efficient algorithm which can be used in other essential kernels such as convolutions, significantly accelerating various deep learning language and vision models.

In this work, we have developed BRGEMM[1] utilising the SVE vector registers to achieve maximum vectorization on Arm, addressing the growing need for e ficient computation on Arm architectures. We chose oneDNN (Deep Neural Network Library) to implement this kernel as it is an open-source performance library which serves as the backend for many popular deep learning frameworks, including PyTorch, TensorFlow, and JAX, making it an ideal platform for implementing optimized algorithms. Our contributions to oneDNN provides a 1.2x to 1.4x performance improvement at the kernel level for various LLM shapes and achieves up to 3x acceleration in inference time for various deep learning language and vision models like Whisper, Resnet50, Llama, T5 etc., in PyTorch on ARM platforms.

This work accelerates high-performance deep learning workload on ARM HPC systems, fostering improved scalability and efficiency for applications ranging from computer vision to NLP and recommendation systems. By developing BRGEMM, we advance the adoption of Arm architecture in AI workloads.

Methodology

BRGEMM : The batch-reduce GEMM kernel performs multiplication on a series of input sub-tensor blocks (batch) and combines the partial results into a single output or accumulator sub-tensor block.[1]

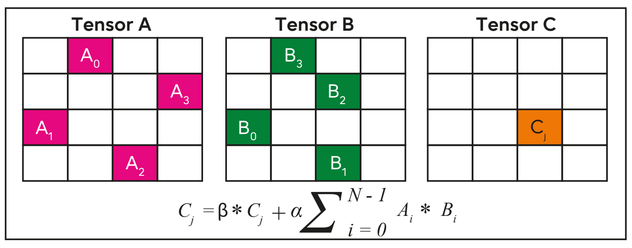

Figure 1 : Batch-reduce operation
The 4 sub-tensors of tensor A are multiplied with the corresponding 4 sub-tensors of B forming a batch and reduced to a sub-tensor of C as show in the formula, where α and β are scaling parameters This reduces load/store operations on C tensor[1].

According to Figure 1 there are 4 matrix multiplications performed in a batch, i.e. A_0xB_0, A_1xB_1, A_2xB_2, A_3xB_3 and the sum of all the results is store in C_j .The Figure 2 shows the GEMM microkernel implementation used by BRGEMM on aarch64 for a single matrix multiplication(e.g. - A_0xB_0) within a batch, written in JIT using Xbyak library in C++. Here for simplicity each vector registers is considered to be 64 bits, which means each vector registers can hold two f32 elements.

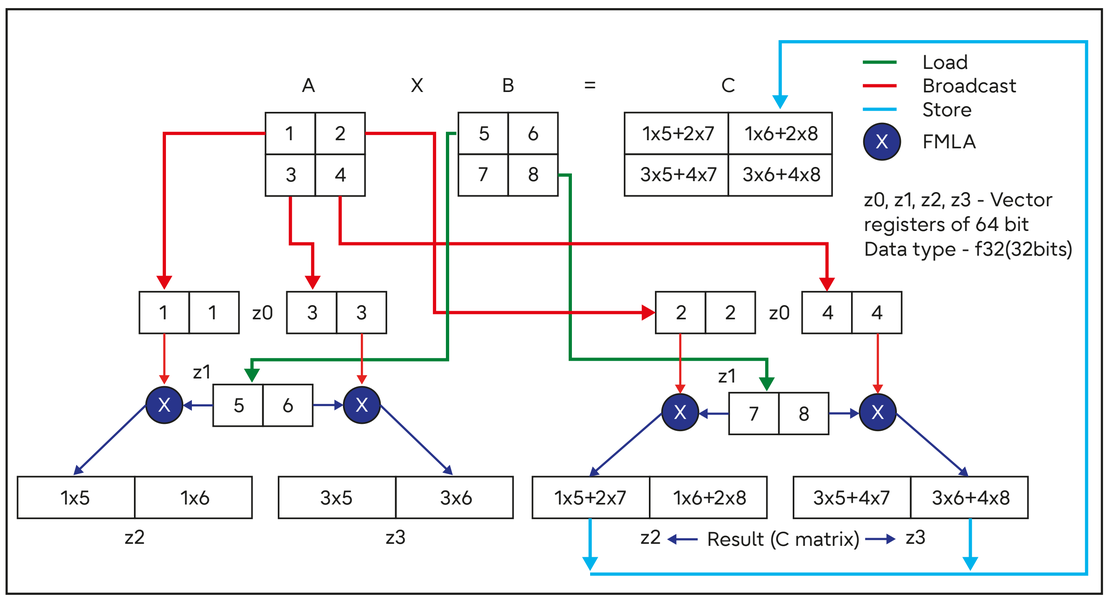

Figure 2 : BRGEMM operation

Matrix A and B are broadcasted and loaded respectively to vector registers. We perform the fused-multiply-add operation(FMLA) on these registers thus storing the results in vector registers termed as accumulators. Once a batch is processed the contents of accumulators are stored in memory assigned to matrix C. The kernel also involves reordering of B matrix into blocked format to enable better memory and cache utilisation.

Convolution in CNNs can be reformulated as BRGEMM-based matrix multiplication with appropriate input and weight transformations.[1]

Figure 3 : BRGEMM Convolution Flow

1 Input tensors and weights are divided into smaller blocks, which are reshaped into matrices—one representing input activations and the other kernel weights. Multiple such flattened pairs are prepared in parallel for efficient processing.

2 For each spatial location in the output, the BRGEMM kernel processes the corresponding weight and input blocks and accumulates the results in the output tensor.

3 The accumulated results from the BRGEMM operations are reshaped back into the final output tensor format.

Thus, the integration of the batch-reduce GEMM kernel removes the necessity for a dedicated convolution kernel. The optimizations applied to the BRGEMM kernel directly contribute to performance improvements in BRGEMM-based convolution.

Results

We have tested performance of our kernel at 2 levels, shape-wise at oneDNN level using the benchdnn and at model inference level using Pytorch on AWS Graviton 3E machine with 32 cores.

The graph in Figure 4 demonstrates the **speed-up (up to 1.4x)** of our BRGEMM kernel compared to the current oneDNN implementation, based on shapes gathered from various language and vision models in PyTorch, including but not limited to Tinyllama, Whisper, DETR, Albert.

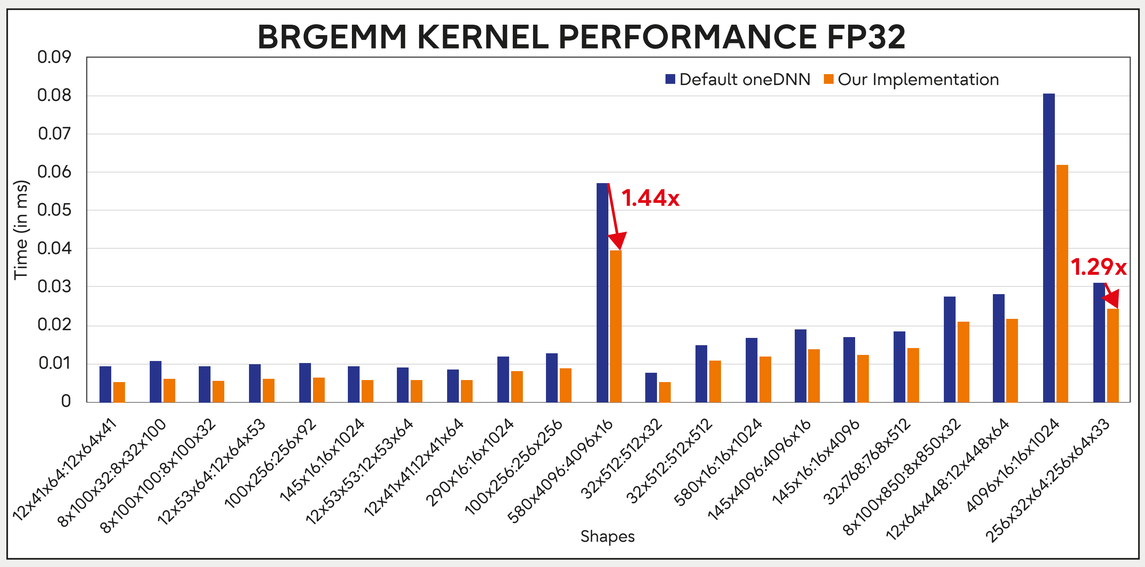

Figure 4 : Shape wise Speed-up

The graph in Figure 5 illustrates the inference **speed-ups (up to 4.5x)** achieved in widely used language and vision models in PyTorch, compared to the current default implementation on ARM.

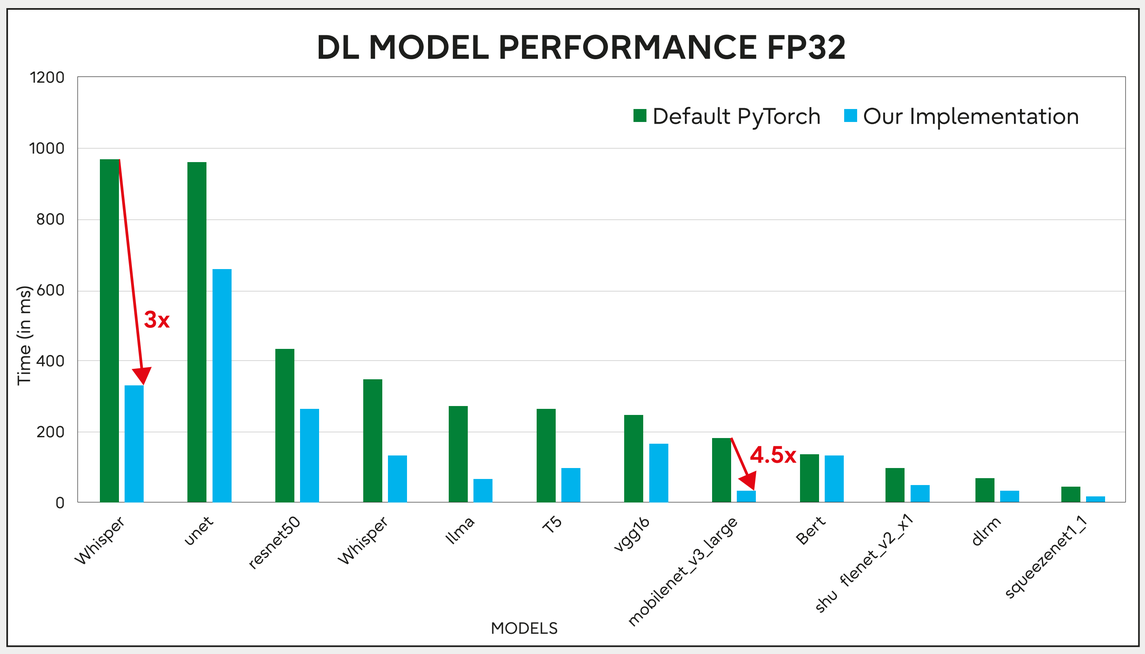

Figure 5 : Inference Speed-up in DL models

The performance acceleration observed on various individual shapes at oneDNN level are effectively extended to model level inference accelerations thus enhancing the performance of deep learning workloads on ARM.

Conclusion and Future Work

- The use of vector registers and data reorders in BRGEMM has effectively accelerated matrix math on ARM CPUs.
- Our contribution has significantly enhanced the performance of deep learning language and vision models like Whisper, Resnet50, Llama, T5 etc., on ARM CPUs.
- Our future work aims to implement quantisation in BRGEMM kernel to accelerate matmuls and convolutions in quantised deep learning workloads.

References

[1] Georganas, Evangelos & Banerjee, Kunal & Kalamkar, Dhiraj & Avancha, Sasikanth & Venkat, Anand & Anderson, Michael & Henry, Greg & Pabst, Hans & Heinecke, Alexander. (2019). High-Performance Deep Learning via a Single Building Block. 10.48550/arXiv.1906.06440.

Brgemm Matmul – A64FX

Brgemm Convolution – A64FX

Brgemm Matmul – GR3

Brgemm Convolution – GR3

Porting OneDNN to Fugaku

This poster is based on results obtained from a project, JPNP21029 subsidized by the New energy and Industrial Technology Development Organization (NEDO).