

Project RP014: Optimizing Llama.cpp and GGML for RVV

Implementation Plan

- **Phase 1: VLEN-agnostic RVV Kernels**
 - **Floating-Point Kernels**
 - SIMD Mappings
 - High-priority Kernels
 - Activation Functions and other Utilities
 - **Llamafile SGEMM**
 - Repacking GEMM and GEMV
 - Quantization Kernels
 - Quantization/Dequantization
 - Vec Dot
 - Repack GEMM and GEMV
 - Llamafile SGEMM Quantization
- **Phase 2: Dynamic Dispatching**
 - Integration
- **Phase 3: Benchmarking and Testing Software**
 - Development
 - Integration with Mainline Llama.cpp

Llamafile SGEMM

Tiled Matrix-Matrix Multiplication library (*ggml-cpu/llamafile/sgemm.cpp*)

Architecture Support

It currently provides matrix-matrix multiplication for the following architectures:

- x86
- ARM
- PowerPC

GGML_OP_MUL_MAT selects:

- **Llamafile_SGEMM** for matrix-matrix
- **Vec_Dot_F16** for matrix-vector

Missing for RVV <- We will be adding this

Data Types

- Floating Point Kernels (FP16 / FP32 / BF16)
- Few Quantization Types (Q8_0, Q4_0, Q5_0, IQ4_NL)

We will be focusing on only floating-point for now

Quantization Types prefer “**Repack GEMM and GEMV**” Kernels

We will work on these later on

Llamafile SGEMM (Tiling)

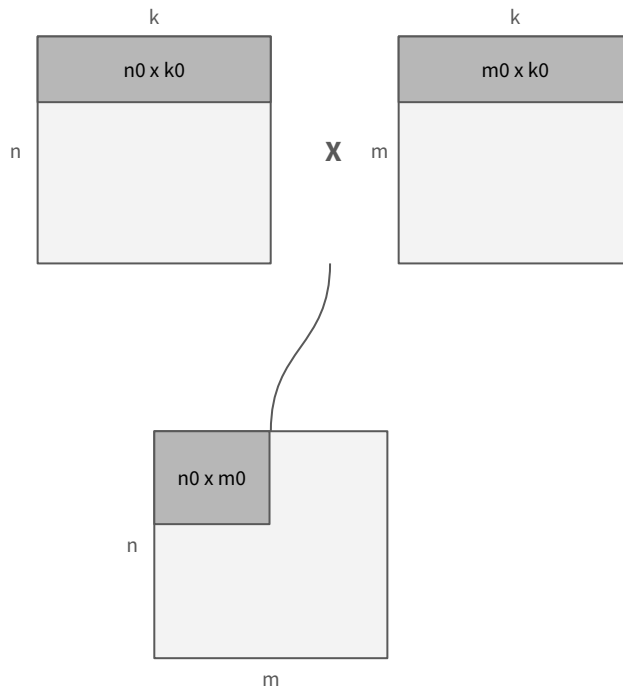
`llamafile_sgemm` implements **different tiling sizes ($n_0 \times m_0$)** based on the number of available registers

- **3x4**, if the architecture supports **16 vector registers**
- **6x4**, if the architecture supports **32 vector registers**

For **RVV**, By **modifying the LMUL**, we can implement multiple tiling schemes:

- **3x4** with **LMUL=1** (32 register groups)
- **6x4** with **LMUL=2** (16 register groups)
- **2x2** with **LMUL=4** (8 register groups)

We will explore tiling combinations under varying LMUL configurations. Based on **benchmarking** results, we will add in the **most effective option**.



Llamafile SGEMM (Job Size)

Job → Number of tiles to process per thread

llamafile_sgemm also adds **job sizes** based on the **input dimensions** and the **number of registers** available.

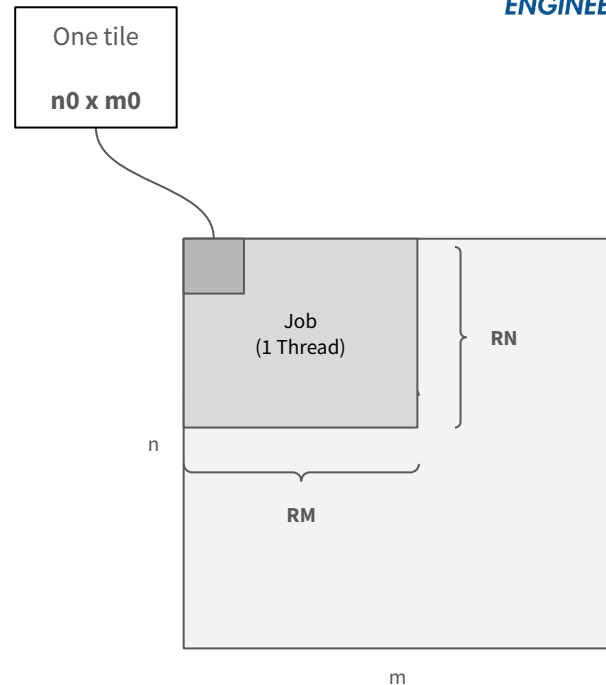
RM = Number of tiles along m dimension

- **RM = 4 if ($m \% 16$)**
- **RM = 2 if ($m \% 8$)**
- **RM = 1 if ($m \% 4$)**

RN = Number of tiles along n dimension

- **RN = 12 (for 32 vector registers)**
- **RN = 24 (for 16 vector registers)**

Based on **benchmarking** results, we will select which job sizes to use.



Llamafile SGEMM (Implementation)

```
class tinyBLAS_RVV {
    bool matmul(int64_t m, int64_t n) {
        ...
    #if LMUL == 1
        if (m % 16 == 0 && (m/16 >= params->nth)) {
            const int64_t SIZE_N = BLOCK_SIZE<6>(n);
            gemm<4, 6, 4>(m, n, SIZE_N, 12);
            return true;
        }
        if (m % 8 == 0) {
            const int64_t SIZE_N = BLOCK_SIZE<6>(n);
            gemm<4, 6, 2>(m, n, SIZE_N, 12);
            return true;
        }
        if (m % 4 == 0) {
            const int64_t SIZE_N = BLOCK_SIZE<6>(n);
            gemm<4, 6, 1>(m, n, SIZE_N, 12);
            return true;
        }
    #elif LMUL == 2
        ...
    #else // LMUL == 4
        ...
    #endif
        return false;
    }
}
```

job_m

gemm<4, 6, 4>(m, n, SIZE_N, 12);

job_n

- job_m = 4 if (m % 16)
- job_m if (m % 8)
- job_m if (m % 4)
- job_n = 12 (for 32 vector registers)
- job_n = 24 (for 16 vector registers)

4x6 with LMUL=1

4x3 with LMUL=2 (16 register groups)

2x2 with LMUL=4 (8 register groups)

Llamafile SGEMM (Implementation)

```
bool llamafile_sgemm(...) {  
    switch (Atype) {  
    case GGML_TYPE_F16: {  
        // Other architectures  
        ...  
        #elif defined(__riscv_v_intrinsic)  
            #if LMUL == 1  
                tinyBLAS_RVV<vfloat32m1_t, vfloat16mf2_t, ggml_fp16_t, ggml_fp16_t, float> tb { ... }  
  
            #elif LMUL == 2  
                tinyBLAS_RVV<vfloat32m2_t, vfloat16m1_t, ggml_fp16_t, ggml_fp16_t, float> tb{ ... }  
  
            #elif LMUL == 4  
                tinyBLAS_RVV<vfloat32m4_t, vfloat16m2_t, ggml_fp16_t, ggml_fp16_t, float> tb{ ... }  
  
    }  
}
```

Llamafire SGEMM (Kernel)

```
template <int RM, int RN, int BM>
    NOINLINE void gemm(int64_t m, int64_t n, int64_t BN) {

    // calculate number of jobs based on tiling and job size

    // parallel region for each thread
    while (job < nb_job) {
        ...
        gemm_bloc<RM, RN>(ii, jj);
        ...
    }

    ...
}
```

Llamafire SGEMM (Kernel)

```
template <int RM, int RN>
inline void gemm_bloc(int64_t ii, int64_t jj) {
    D Cv[RN][RM] = {}; // Accumulators based on tile size

    for (int64_t l = 0; l < k; l += KN) { // Iterate over full rows
        // Load all rows of B
        V Bv[RN];
        for (int64_t j = 0; j < RN; ++j) {
            Bv[j] = load<V>(B + ldb * (jj + j) + 1);
        }

        // Load 1 row of A and accumulate
        for (int64_t i = 0; i < RM; ++i) {
            V Av = load<V>(A + lda * (ii + i) + 1);
            for (int64_t j = 0; j < RN; ++j) {
                Cv[j][i] = madd(Av, Bv[j], Cv[j][i]);
            }
        }
    }

    // Reduce
    for (int64_t j = 0; j < RN; ++j)
        for (int64_t i = 0; i < RM; ++i)
            C[ldc * (jj + j) + (ii + i)] = hsum(Cv[j][i]);
}
```

SGEMM F16 Tiling

Tiles	Accumulator (F32) Widened			Inputs (F16)			Total
	Total	LMUL	Registers	LMUL	A	B	Registers
4x6	24	1	24	1/2	1	6	31
4x3	12	2	24	1	3	1	28
2x2	4	4	16	2	1	4	26

SGEMM Benchmarking Strategy

Goal: Find the most performant tiling and job size

Benchmarking Parameters:

- Typical Input Sizes in TinyLlama and BERT
- LMUL Sweeps (LMUL=1, 2 and 4) mapping to Tiling (4x6, 4x3, 2x2)
- 64-byte alignment
- Performance with multiple threads

Methodology:

- Run a high number of iterations per SGEMM matmul on all threads
- Find the best time from the iterations

Performance matrix:

- Best time (us)

Using **test-backend-ops.cpp** (in Llama.cpp)

SGEMM F16 - Benchmarking

Type	Input Sizes			Time (us)			
	M	N	K	4x6 (LMUL = 1)	4x3 (LMUL = 2)	2x2 (LMUL = 4)	vec_dot
F16	64	32	256	54	35	33	50.21
F16	256	32	64	52	45	42	71.13
F16	256	32	2048	3601	2794	3190	1670
F16	2048	32	2048	29383	20681	26270	12499
F16	2048	32	5632	19450	19353	30519	79335
F16	5632	32	2048	76941	67720	69659	31849
F16	64	128	256	181	151	152	186
F16	256	128	64	176	177	168	253
F16	256	128	2048	16668	10263	7418	6278
F16	2048	128	2048	106849	76419	55729	56275
F16	2048	128	5632	100893	76512	104004	305193
F16	5632	128	2048	307912	204202	153366	124972
F16	64	512	256	760	647	642	832
F16	256	512	64	755	692	662	972
F16	256	512	2048	69358	42745	26916	26463
F16	2048	512	2048	454796	312554	231930	187110
F16	2048	512	5632	376643	312758	376550	1223595
F16	5632	512	2048	1217249	776807	2485170	502829

SGEMM Benchmarking Strategy

Goal: Find the most performant tiling and job size

Benchmarking Parameters:

- Typical Input Sizes in TinyLlama and BERT
- LMUL Sweeps (LMUL=1, 2 and 4) mapping to Tiling (4x6, 4x3, 2x2)
- 64-byte alignment
- Performance with multiple threads
- Job Sizes

Methodology:

- Run a high number of iterations per SGEMM matmul on all threads
- Find the best time from the iterations

Performance matrix:

- Best time (us)

SGEMM F16 - Benchmarking

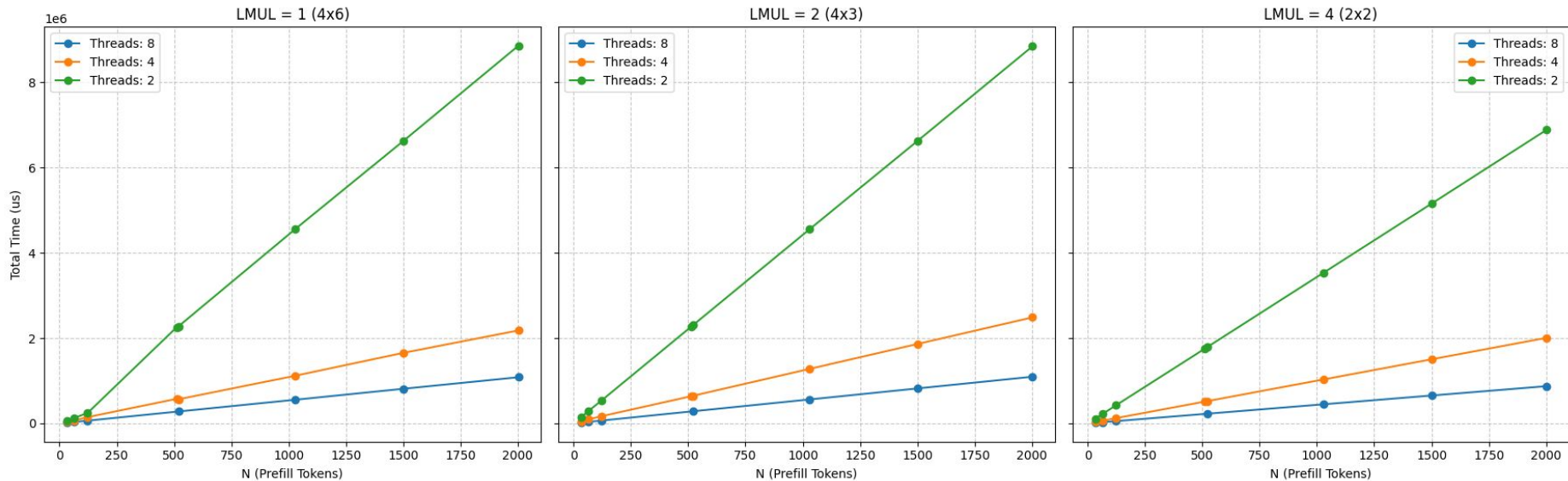
Type	M	N	K	JOB_M	JOB_N	Threads	4x6 (LMUL=1)	4x3 (LMUL=2)	2x2 (LMUL=4)
							Total Time	Total Time	Total Time
F16	64	32	256	4	4	8	67	53	47
F16	256	32	64	4	4	8	35	46	42
F16	256	32	2048	4	4	8	2173	2041	1745
F16	2048	32	2048	4	4	8	16956	16250	13918
F16	2048	32	5632	4	4	8	64059	47644	45200
F16	5632	32	2048	4	4	8	46740	44754	38611
F16	64	64	256	4	4	8	147	95	83
F16	256	64	64	4	4	8	100	92	84
F16	256	64	2048	4	4	8	4911	4080	3556
F16	2048	64	2048	4	4	8	38864	32599	27780
F16	2048	64	5632	4	4	8	139532	95864	90545
F16	5632	64	2048	4	4	8	106311	89331	77520
F16	64	128	256	4	4	8	281	187	157
F16	256	128	64	4	4	8	192	183	163
F16	256	128	2048	4	4	8	9692	7978	6962
F16	2048	128	2048	4	4	8	77755	64100	55649
F16	2048	128	5632	4	4	8	280462	192255	180948
F16	5632	128	2048	4	4	8	213160	176006	155001

SGEMM F16 - End-to-End Performance

MILK-V Jupiter (Spacemit X60)					
Model	Prefill Prompt Size	Tokens / second (avg)			
		Llamafile SGEMM			Vec Dot
		4x6	4x3	2x2	
Tinyllama F16 1.1B	32	6.08	7.89	6.26	8.42
Tinyllama F16 1.1B	64	6.09	7.25	11.31	7.57
Tinyllama F16 1.1B	128	5.93	6.9	13.73	8.78
Tinyllama F16 1.1B	256	5.54	6.79	12.56	8.57
Tinyllama F16 1.1B	512	5.37	6.64	13.37	8.68

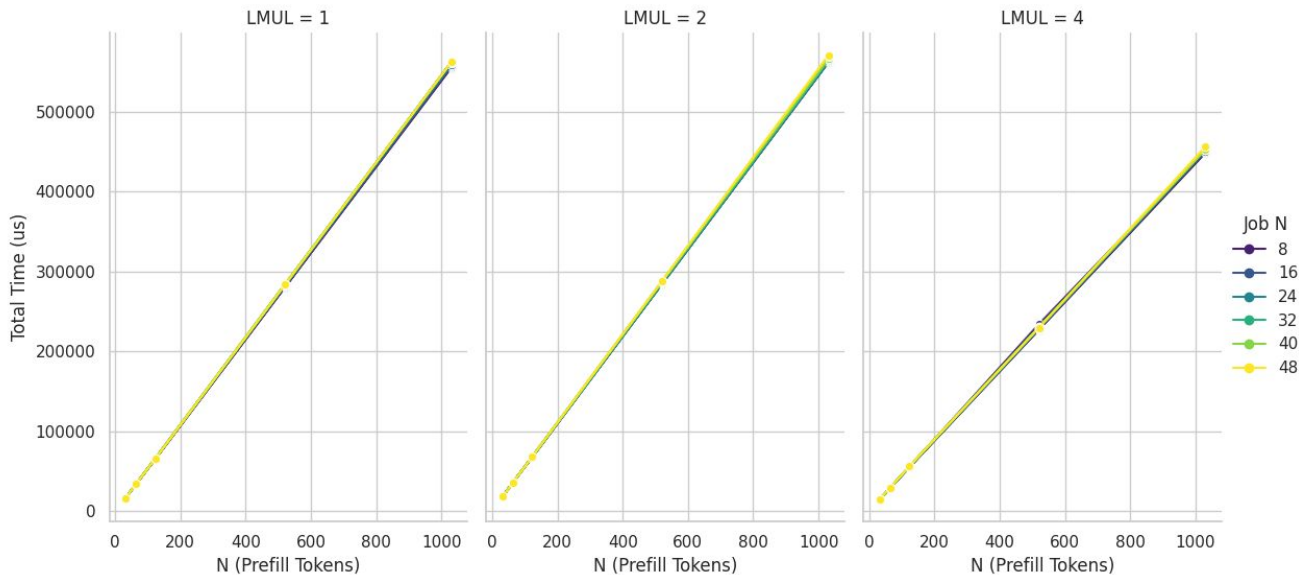
Llamafile SGEMM - Performance Plots

N vs Total Time (Constraints: M=2048, K=2048, Job N=4)



Llamafile SGEMM - Performance Plots

- No effect of N dim Job size on performance
- LMUL=4 (2x2) performs best



Llamafile SGEMM - Prompt Processing

BananaPI BPI-F3 (Spacemit X60)					
Model	Prompt Size	Tokens / second (avg)			
		Llamafile SGEMM			Vec Dot
		4x6	4x3	2x2	
Tinyllama F16 1.1B	32	6.08	7.89	6.26	8.42
Tinyllama F16 1.1B	64	6.09	7.25	11.31	7.57
Tinyllama F16 1.1B	128	5.93	6.9	13.73	8.78
Tinyllama F16 1.1B	256	5.54	6.79	12.56	8.57
Tinyllama F16 1.1B	512	5.37	6.64	13.37	8.68