BUILDING NEW GGML BACKENDS

FOR NOVEL ACCELERATORS, HOW, CHALLENGE AND OPPORTUNITIES

Marin Chang

2025/02/02 @ FOSDEM Low level AI engineering track

DISCLOSURE

I am sponsored by Tenstorrent through support grants and hardware access.

This talk may contain biases, and Tenstorrent had no involvement in its development or review. I've made an effort to ensure accuracy and objectivity. The work presented started wihtout and would happen with or without the sponsorship.

BACKGROUND

#WHOAMI

- C++ and HPC
- FOSS developer
- My version of fun

WHAT I DO (UNRELATED TO THIS TALK)

- drogon - fast C++ web framework
- tlgs - Gemini protocol search engine
- landlock-unveil - OpenBSD unveil for Linux

GGML: EFFICIENT INFERENCE

- Efficient inference, especially for LLMs
- Strong quantization support
- Flexible, growing community
- Ideal for exploring new hardware

MY JOURNEY: EFFICIENT AI

- Late 2022: LLMs were new, RK3588 was exciting
- Goal: Local AI assistant without the heat
- Got RK3588 NPU working in llama.cpp, but it wasn't suitable for LLMs
- Discovered Tenstorrent and their Grayskull e75

TENSTORRENT: NOT A GPU

- Manycore: RISC-V + tensor engines
- "Baby" RISC-V cores for control
- Grid of cores, Network-on-Chip (NoC)
- No global cache or unified memory
- Explicit data flow management
- Scales by adding more chips

Big RISC-V & Baby RISC-V

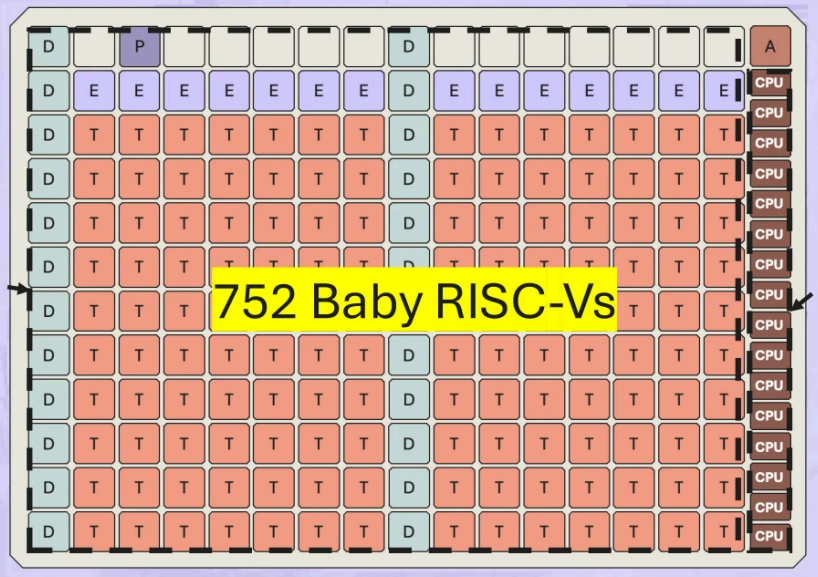
Baby RISC-V

Feature	Spec
Total Baby RISC-Vs	752
Compute	32-bit Int multiplier / divider Floating point (FP32 / BFLOAT16) 128-bit vector (1 per Tensix)
I-cache	4 KB
D-scratch	8 KB

T Tensix cores

D DRAM cores

E ETH cores



Big RISC-V

Feature	Spec
RISC-V CPUs	x16 (4 clusters of 4)
Compute	64-bit, dual-issue, in-order
L3 cache	2 MB / CPU
L2 cache	128 KB / CPU
L1 I-cache	32 KB / CPU (2 way associative)
L1 D-cache	32 KB / CPU (4 way associative)

16 Big RISC-Vs

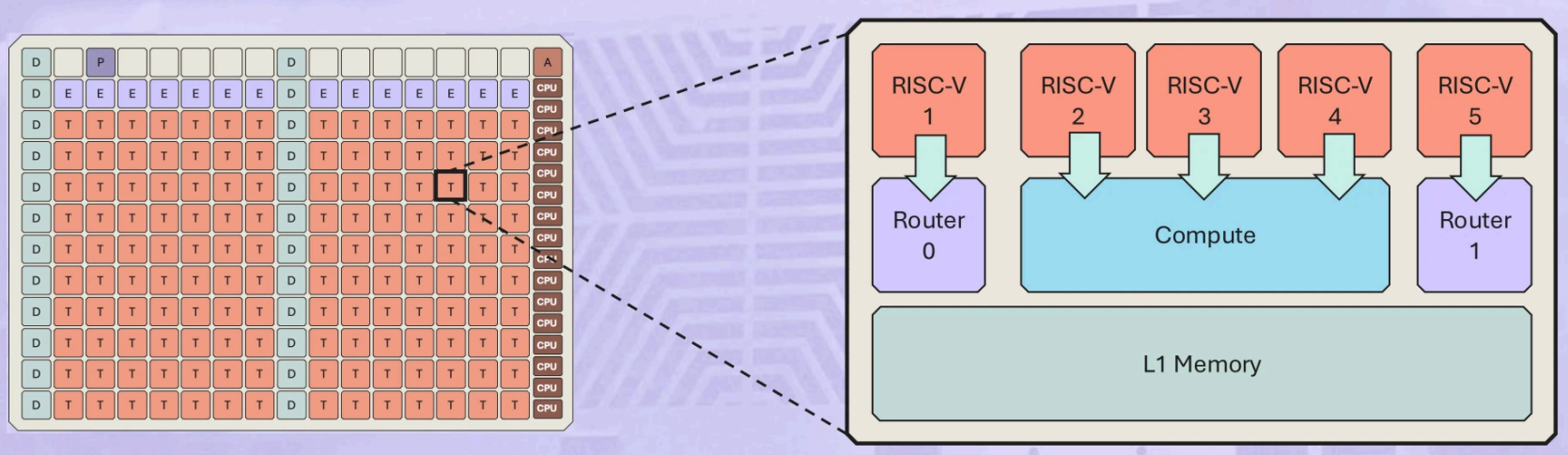
- Runs Linux
- On-device host for the AI accelerator

C RISC-V CPUs

source: Tenstorrent Hotchips 2024 slides

All RISC-V Programmable
Within the Tensix Core

- 5 baby RISC-Vs
- 32-bit RISC-V ISA

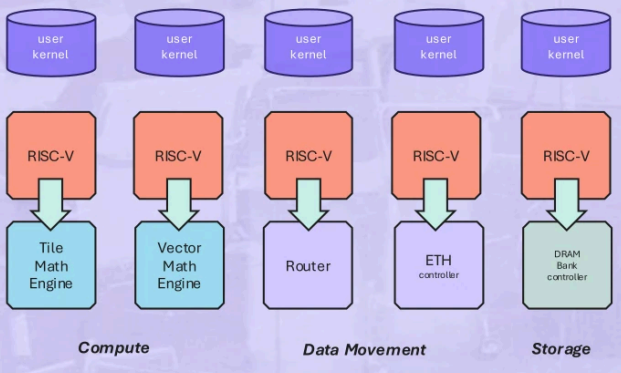


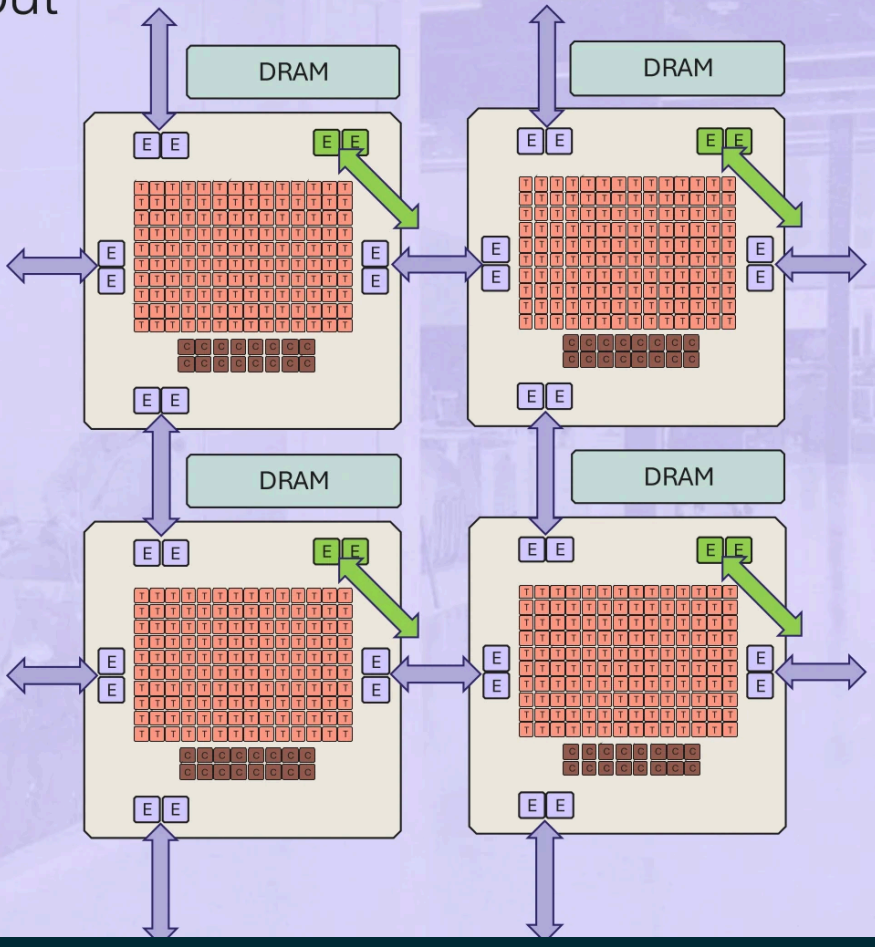
source: Tenstorrent Hotchips 2024 slides

Blackhole: Ethernet-Based Scale out

- 1 TB/s of Blackhole Ethernet
- Can be connected into any topology

- Mesh topology is great for AI
  - Locality and regularity of data movement
  - Sharded data
  - 200 GB/s in N / S / W / E / Z
  - 2D / 3D torus





source: Tenstorrent Hotchips 2024 slides

TENSTORRENT SDKS

TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

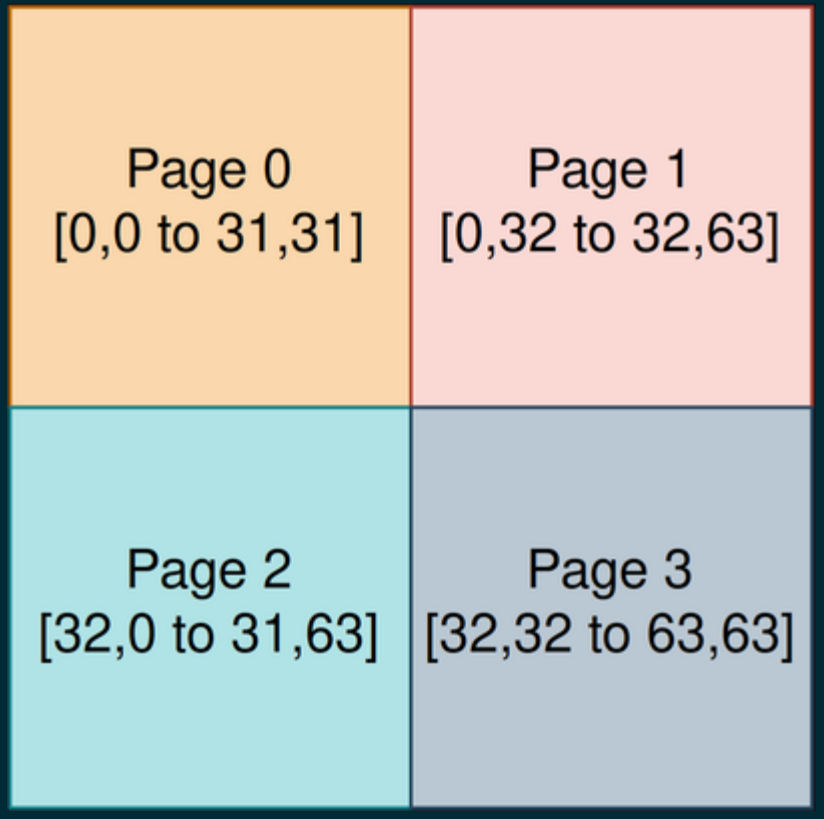
TTNN EXAMPLE

```
1 int main() {
2     auto device = &ttnn::open_device(0);
3     auto a = ttnn::arange(0, 100, 1,
4         tt::tt_metal::DataType::BFLOAT16);
5     a = ttnn::tilize_with_zero_padding(a.to(device));
6     auto res = ttnn::multiply(a, 2.f);
7     std::cout << res.write_to_string() << std::endl;
8 }
```

TILED TENSORS & DATA

- Tiled layout for efficiency
- bfloat16 with optional float32
- Custom types: `BFLOAT8_B`, `BFLOAT4_B`

TILES


Page 0
[0,0 to 31,31]
Page 1
[0,32 to 32,63]
Page 2
[32,0 to 31,63]
Page 3
[32,32 to 63,63]

L1, DRAM

- Tenstorrent calls it'sSRAM scratchpad "L1"
- Not cache
- You can strore tensors there
- High bandwidth
- DRAM is slow but huge (100MB vs 12GB)

BUILDING NEW BACKENDS

- Register backend
- Scan for supported ops
- Accept tensor from GGML
- Execute operations
- Give GGML results on request

BRIDGING: GGML & TTNN

- GGML: row-major, TTNN: tiled
- Metalium: no device memory mapping
- Typed allocation mismatch
- Quantization type differences

GGML BACKEND REGISTRATION

- ggml_backend_registry holds all backends

```
struct ggml_backend_registry {
    std::vector backends;
    std::vector devices;

    ggml_backend_registry() {
#ifdef GGML_USE_CUDA
        register_backend(ggml_backend_cuda_reg());
#endif
#ifdef GGML_USE_METAL
        register_backend(ggml_backend_metal_reg());
#endif
#ifdef GGML_USE_SYCL
        register_backend(ggml_backend_sycl_reg());
#endif
#ifdef GGML_USE_VULKAN
```

GGML BACKEND REGISTRATION

```cpp
ggml_backend_reg_t ggml_backend_metalium_reg() {
  static ggml_backend_reg reg; static std::once_flag once;
  std::call_once(once, [&]() {
    auto ctx =
      std::make_unique<ggml_backend_metalium_reg_context>();
    // initialize and collect device info here
    ...
    reg = ggml_backend_reg {
      /*.api_version = */GGML_BACKEND_API_VERSION,
      /*.interface   = */ggml_backend_metalium_reg_interface,
      /*.context     = */ctx.get()
    };
  });
  return &reg;
}
```

GGML BACKEND REGISTRATION

```
ggml_backend_reg_t ggml_backend_metalium_reg() {
  static ggml_backend_reg reg; static std::once_flag once;
  std::call_once(once, [&]() {
    auto ctx =
      std::make_unique<ggml_backend_metalium_reg_context>();
    // initialize and collect device info here
    ...
    reg = ggml_backend_reg {
      /*.api_version = */GGML_BACKEND_API_VERSION,
      /*.interface   = */ggml_backend_metalium_reg_interface,
      /*.context     = */ctx.get()
    };
  });
  return &reg;
}
```

GGML BACKEND REGISTRATION

```cpp
ggml_backend_reg_t ggml_backend_metalium_reg() {
  static ggml_backend_reg reg; static std::once_flag once;
  std::call_once(once, [&]() {
    auto ctx =
      std::make_unique<ggml_backend_metalium_reg_context>();
    // initialize and collect device info here
    ...
    reg = ggml_backend_reg {
      /*.api_version = */GGML_BACKEND_API_VERSION,
      /*.interface   = */ggml_backend_metalium_reg_interface,
      /*.context     = */ctx.get()
    };
  });
  return &reg;
}
```

GGML BACKEND REGISTRATION

- descriptor: data pointer, vtable

```
1 static const ggml_backend_reg_i ggml_backend_metalium_reg_inter
2 /* .get_name           = */ ggml_backend_metaliium_reg_get_name,
3 /* .get_device_count   = */ ggml_backend_metalium_reg_get_device
4 /* .get_device         = */ ggml_backend_metalium_reg_get_device
5 /* .get_proc_address   = */ NULL,
6 };
```

GGML BACKEND REGISTRATION

- descriptor: data pointer, vtable

```
1 static const ggml_backend_reg_i ggml_backend_metalium_reg_inter
2 /* .get_name           = */ ggml_backend_metaliium_reg_get_name,
3 /* .get_device_count   = */ ggml_backend_metalium_reg_get_device
4 /* .get_device         = */ ggml_backend_metalium_reg_get_device
5 /* .get_proc_address   = */ NULL,
6 };
```

TENSOR MANAGEMENT

- GGML: 2 memory pools. Expects MMIO
- Solution: dummy address
- metadata in `tensor->extra`
- Allocate in `set_tensor`
- Deallocate: `buffer_reset`
- Access metatdata in operatiors

DATA IN/OUT

- **set_tensor:**
    1. Dequantize to FP32.
    2. To TTNN tensor.
    3. Upload to device.
    4. Tilize, cast to type.
- **get_tensor:** Reverse.

ACCEPT/REJECT OPERATIONS

```
bool ggml_backend_metalium_device_supports_op_internal
(ggml_backend_dev_t device, const struct ggml_tensor * op) {
    // return true if op is supported on the device
    // Can reject for whatever reason
    switch(op->op) {
    case GGML_OP_NONE: // Tensor data
        return op->type == GGML_TYPE_F32; // float32 only
    case GGML_OP_ADD:
        return ggml_n_dims(op) == 1; // 1D adds only
    default:
        return false;
    }
}
```

RUNNING OPERATIONS

```
enum ggml_status ggml_backend_metalium_graph_compute
    (ggml_backend_t backend, struct ggml_cgraph * cgraph) {
    for (int i = 0; i < cgraph->n_nodes; i++) {
        // ... (switch for ops) ...
        case GGML_OP_ADD:
            ggml_backend_metalium_add(dst);
            break;
    }
    return GGML_STATUS_SUCCESS;
}
void ggml_backend_metalium_add(struct ggml_tensor * dst) {
    ...
    dst->extra->tensor = ttnn::add(dst->src[0]->extra->tensor,
                                   dst->src[1]->extra->tensor);
}
```

Pesudo code. Real one is more complex

DEALING WITH VIEWS

- GGML strides: don't work with tiles.
- Initial: Eager evaluation.
- Now: Lazy evaluation (on demand).
- avoids write/copy issues.

MISSING FEATURES IN GGML

MISSING: GRAPH REWRITE

- No graph rewrite = missed optimizations
- No fusing activations to matmul
- Hard to use on-chip SRAM without it
- Can't do convolution
- List goes on

MISSING: PARALLEL UPLOAD

- Weight upload is single-threaded.
- Dequantize takes time
- Parallelization: overhead for small models.
- Model-level parallelization is ideal.

DEBUGGING TRICKS

- Dimensions are REVERSED in GGML
- `tensor->ne` (shape), `tensor->nb` (stride)
- Stride can be blocked (32 elem in 18 bytes)
- Use many asserts for data consistency
- `llama-eval-callback`: prints tensor
- `diff -y` with CPU result

RESULTS

- Smaller LLMs run on Grayskull/Wormhole.
- Many ops missing, CPU fallback.
- It works (Performance not great yet)
- Critical ops missing

Model	Quant	TTNN quant	tok/s
LLaMA 3.2 8B	Q4_K_M	BFLOAT8_B	3.56
LLaMA 3.1 1B	Q4_K_M	BFLOAT8_B	22.18
TinyLLaMA 1.1B	Q4_0	BFLOAT8_B	21.85
Gemma2 2B	Q4_K_M	BFLOAT8_B	11.14
Gemma2 2B	Q4_K_M	BFLOAT4_B	11.40

On QuietBox + 1 Wormhole chip - as of Dec 2024

FUTURE WORK

- Missing critical operators in TTNN
  - GET_ROWS
  - MUL_MAT/SOFTMAX broadcast
  - FLASH_ATTN_EXT
- Operator expections mismatch
- Device clustering support
- Upstream to llama.cpp

CONCLUSION

- New backends for [A-Z]PUs are very doable
- Given work and hiding the hardware details
- GGML needs new passes

THANK YOU!

- Code: marty1885/llama.cpp @ metalium-support
- Detailed post: On clehaxze.tw
- Reach me at:
  - `@clehaxze:matrix.clehaxze.tw`
  - Discord: `marty1885`