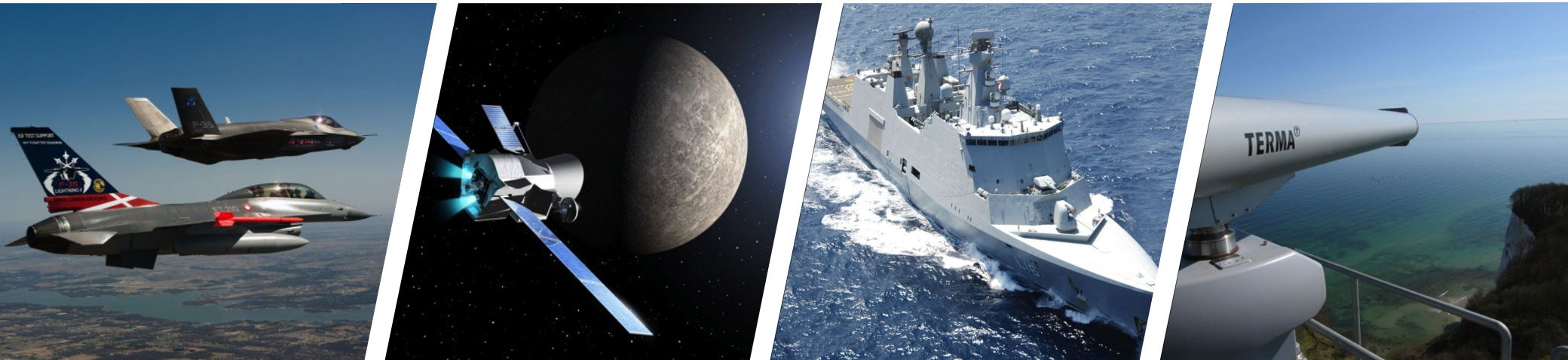
TERMA
ALLIES IN INNOVATION

T-EMU 2.0: Next Generation Emulator

The Terma Emulator Evolution

Dr. Mattias Holm <maho@terma.com>



- **Outline**
  - **Emulation in General**
  - **State of the Art**
  - **T-EMU 2.0**
    - **The Emulator Core**
    - **Memory System**
    - **The Emulator API**
    - **Performance**
  - **Future Directions**

Emulation in General

- **Micro-processor simulator**
  - **Instruction level simulation**
  - **Interpretation**
    - **Decode-dispatch**
    - **Threaded**
  - **Binary translation**
  - **Virtualisation**
    - **Normally no timing accuracy**
- **Memory simulator**
  - **Memory attributes (breakpoint, SEU, etc)**
  - **Virtual memory (MMU)**
  - **Access times**
- **Device models**
  - **Not related to the emulator core directly**
  - **Integrated with memory models**
  - **Need at minimum: MMIO events, timed events, ability to raise IRQs and ability to write to memory**

State of the Art

- **Binary translators**
  - **OVPSim**
  - **Windriver Simics (~350 MIPS)**
  - **QEMU (partially GPL)**
  - **SimLEON**
- **Interpretation (SPARC emulators)**
  - **TSIM (~60 MIPS)**
  - **ESOC Emulator (65 MIPS no MMU, 25 MIPS with MMU)**
  - **T-EMU 2.0…**

T-EMU: The Terma Emulator

- **T-EMU 1:**
  - **Derivation of ESOC Emulator Suite 1.11**
  - **Formed the baseline for the work on ESOC Emulator Suite 2.0**
  - **Written in EMMA: The Extensible Meta-Macro Assembler (embedded assembler, using Ada as host language)**
- **T-EMU 2:**
  - **Complete rewrite**
  - **Using modern C++11 and LLVM**
  - **LLVM compiler tools are used extensively**
  - **Interpreted, but ready to upgrade with binary translation capabilities**
  - **Significant work spent on defining a device modelling APIs**
    - **Can easily be wrapped for scripting languages (e.g. prototype your device model in Python) or SMP2**
  - **Can emulate multi-core processors**

- **Major Areas of Focus**
  - **Emulator Core**
    - **Instruction level simulation with static timing**
    - **Designed for performance and flexibility**
  - **Memory**
    - **Memory spaces and memory accesses**
    - **Ability to insert both statistical and exact cache models if needed.**
  - **Device Models**
    - **Standard devices used in the European space sector:**
      - **MEC, LEON2, GRLIB (UARTs, timers, interrupt controllers etc)**
  - **End user APIs**
    - **Physical address independent devices**
    - **Automatic checkpointing**
    - **Automatic access to device properties from scripts**
    - **C interface**

T-EMU 2.0: Features and Models

- **Library based design**
  - **Easy to integrate in simulators**
- **Command Line Interface**
  - **Assisting with emulator and model development and integration**
  - **On-board software development (e.g. unit tests)**

T-EMU 2.0: Features and Models

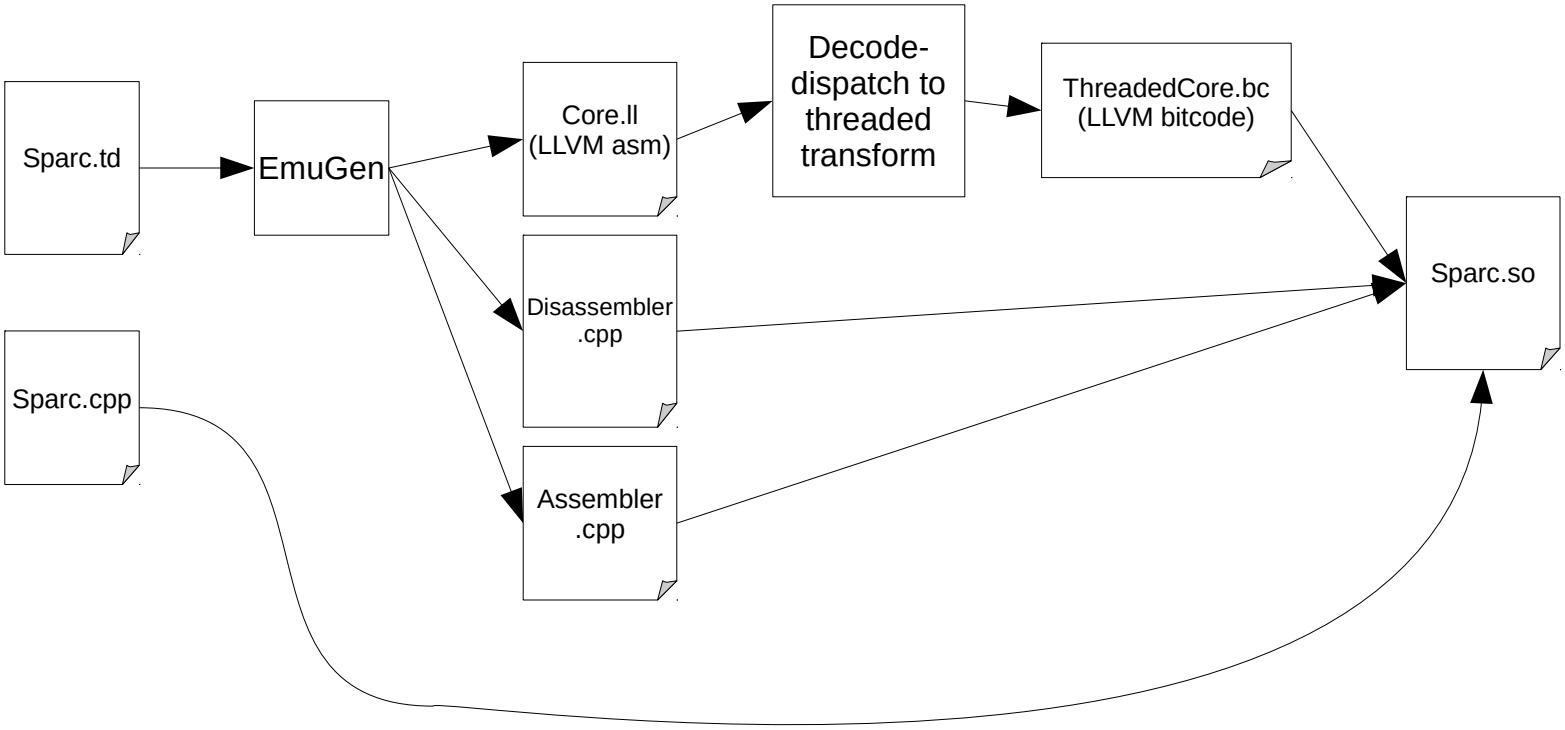
- **Processors**
  - **ERC32**
  - **LEON2**
  - **LEON3**
  - **LEON4**
  - **NOTE: SMP and multi-core processors can be emulated.**
- **Models**
  - **On-Chip Devices**
    - **MEC (ERC32)**
    - **LEON2 on-chip devices**
  - **GRLIB:**
    - **AHBCTRL, AHBSTAT, AHBUART**
    - **APBCTRL, APBUART, FTMCTRL**
    - **GPTIMER, IRQMP**
    - **Additional added as we go along.**
- **Buses**
  - **AMBA**
    - **PNP supported for AMBA devices**
  - **Serial**
  - **More bus-models to be added**

- **Emulator Cores:**
  - **Written in TableGen and LLVM assembler**
  - **(Operational) decode-dispatch cores transformed to threaded code automatically using custom LLVM transformation passes.**
  - **TableGen data combines: instruction decoders, instruction semantics and assembler syntax in a transformable format**
  - **Multi-core support**
- **Emulator Shell**
  - **Implemented using the T-EMU 2.0 object system APIs**
  - **Integrates auto-generated assemblers and disassemblers generated from TableGen data.**
  - **High level interfaces**
    - **Interrupt interface, memory interface, etc**

T-EMU 2.0: TableGen CPU Descriptions

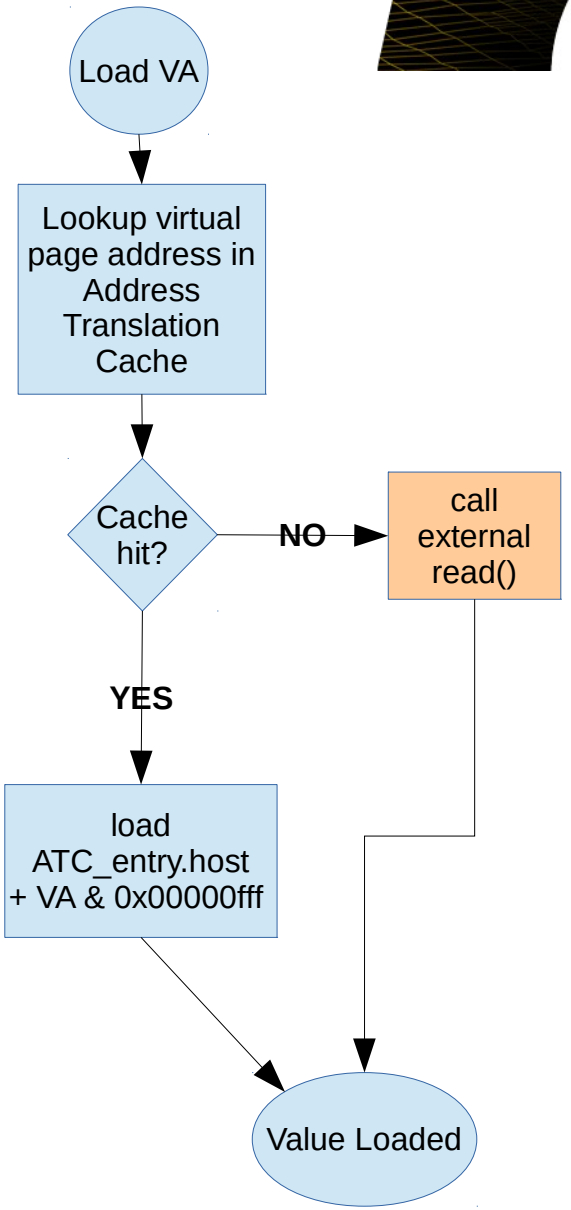
```
def add_rr : fmt3_1 <0b10, 0b1010101> {
  let AsmStr = “add {rs1:gpr}, {rs2:gpr}, {rd:gpr}”;
  let Semantics = [{
    %r1 = call i32 @emu.getReg(%cpu_t* %cpu, i5 %rs1)
    %r2 = call i32 @emu.getReg(%cpu_t* %cpu, i5 %rs2)
    %res = add i32 %r1, %r2
    call void @emu.setReg(%cpu_t* %cpu, i5 %rd, i32 %res)
  }];
};
```

T-EMU 2.0: Compilation Pipeline



T-EMU 2.0: Memory Emulation

- **Each processor has a memory space attached to it:**
  - **Memory space decodes addresses**
- **N-level page table for identifying memory mapped objects**
  - **memory**
  - **devices**
- **Unified interface for memory and devices:**
  - **Memory Access Interface**
  - **Zero-overhead for MMU due to address translation cache**
- **Memory attributes**
  - **breakpoint, watchpoint read + write, upset, faulty, user 1,2,3**



T-EMU 2.0: Application Programming Interface

- **Driven by the object system:**
  - **Classes**
    - **Description for how to create objects and what fields an object has**
  - **Properties**
    - **Fields in a class with optional read and write functions**
      - **Used for register emulation**
    - **Named, can access by string names (useful in scripts)**
  - **Interfaces**
    - **Structs of function pointers registered with a class**
      - **Named, can query dynamically by name**
      - **Normally queried at machine configuration time**
      - **Interfaces are cached in fat pointers**
    - **Used for memory accesses (a device implements the memory access interface)**
  - **Objects**
    - **Instances of a class**
      - **Objects are named, scripts can query for a given object, e.g. cpu0**

T-EMU 2.0: Device Models

MMIO Models Implement the MemAccessIface:

```
typedef struct temu_MemAccessIface {
  void (*fetch)(void *Obj, temu_MemTransaction *Mt);
  void (*read)(void *Obj, temu_MemTransaction *Mt);
  void (*write)(void *Obj, temu_MemTransaction *Mt);
} temu_MemAccessIface;
```

The functions take a potiner to a MemTransaction object (which is constructed by the core):

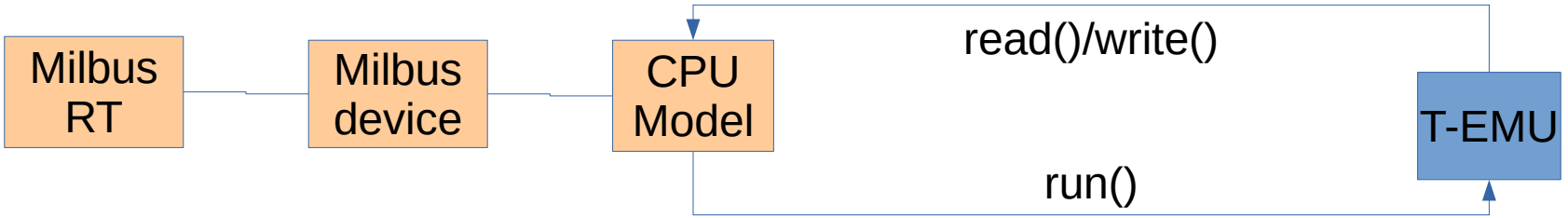
```
typedef struct temu_MemTransaction {
  uint64_t Va; // Virtual addr
  uint64_t Pa; // Physical addr
  uint64_t Value; // Out or in value
  uint8_t Size; // Log size of access

  uint64_t Offset; // Pa – Dev Start
  void *Initiator; // CPU pointer
  void *Page; // Out (for ATC)
  uint64_t Cycles; // Out (cost of op)
} temu_MemTransaction;
```

T-EMU 2.0: Application Programming Interface

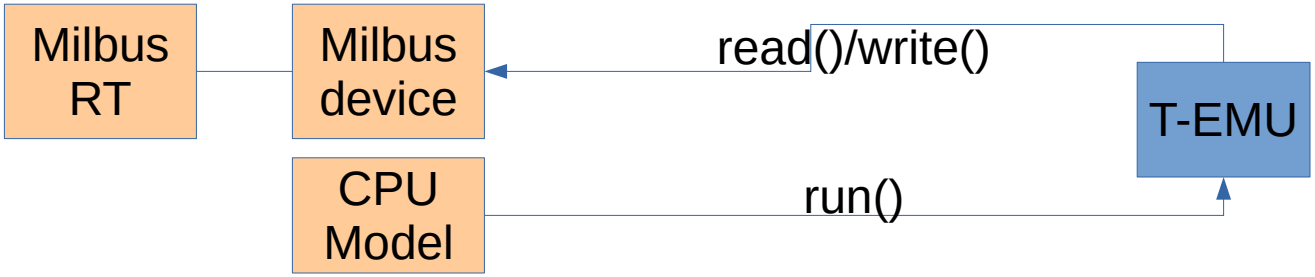
- **SMP2 Support**
  - **Models can be integrated with SMP2 based simulators if needed.**
  - **Similar to how ESOC Emu and TSIM are integrated in an SMP2 environment today.**
  - **Similarities between T-EMU 2.0 object system and SMP2, integration is straight forward.**
- **Other modelling frameworks to integrate with:**
  - **System-C models**
  - **VHDL / Verilog**
- **Emulator should integrate with any external API!**
- **Models can be prototyped in scripting languages (e.g. Python)**

T-EMU 2.0: SMP2 Options

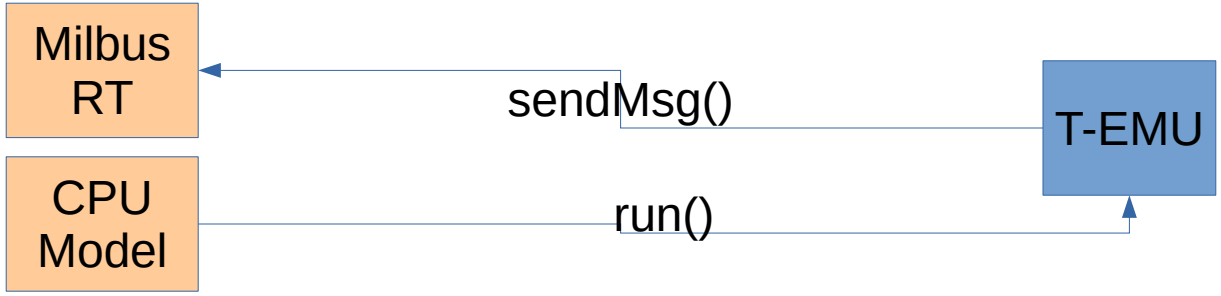
Traditional



Traditional ++



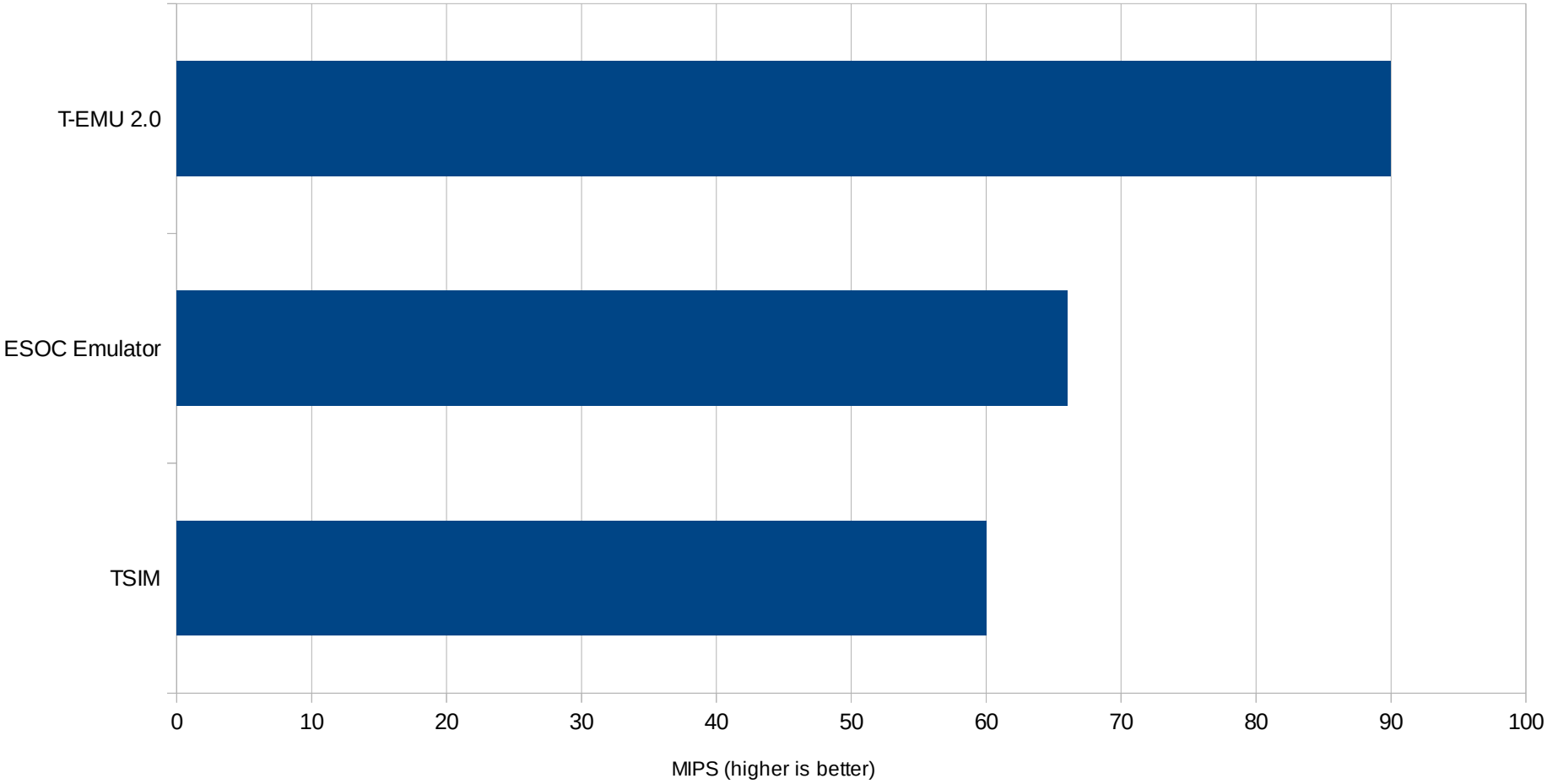
Ideal for Emulator



T-EMU 2.0: Correctness and Accuracy

- **Instruction level correctness**
  - **No-pipeline dependent effects at present**
  - **E.g. nops after WRPSR not detected**
- **Timing accuracy**
  - **Static instruction timing**
  - **Static memory access penalty (waitstates)**
  - **Caches can be added (but exact cache models imply that the ATC cannot be used to its full extent)**
    - **Exact cache models have performance implications**
    - **T-EMU 2.0 allows the use of cache models if needed, but for high performance emulation the user would simply omit connecting them to the CPU.**
      - **Lack of cache model in simulated system = no performance impact of supporting the feature.**
      - **Add cache model, pay the penalty**
  - **T-EMU 2.0 scales between different timing accuracy levels:**
    - **Low accuracy (no cache models, static waitstates only), highest performance**
    - **Medium accuracy (statistical cache model assigning a cache miss penalty per page)**
    - **High accuracy (exact cache models), lower performance (cache model invoked on every fetch, read and write)**
  - **Option:**
    - **Run at high accuracy for profile used in a medium accuracy statistical model**

Current Emulator Performance



- 3.5 GHz x86-64
- ESOC Emu can be squeezed a bit more using custom TERMA optimisations, numbers here are for the stock ESOC Emu configuration. Current ongoing optimisation work.
- TSIM numbers from Gaisler's website.
- Anything above 50 MIPS is high performance for an interpreted emulator

T-EMU 2.0: Future Directions

- **Built-in support for SMP2 scheduler (some simulators may want to have centralised scheduling so that an interrupt is raised by a model is detected directly instead of being delayed to the next emulator scheduling).**
- **Language for device modelling**
  - **API compatibility between different emulators can be difficult**
  - **DSL for writing device models could use compiler techniques (front and backends) to generate models for multiple emulators.**
- **Binary translation (>300 MIPS)**
- **Additional architectures (ARM, PowerPC, MIPS etc)**
- **Support more ways for device modelling:**
  - **SMP2 publication**
  - **System-C**
- **Bigger model library:**
  - **Provide models for all common spacecraft processors and peripherals**
  - **More GRLIB models**
  - **Other models (e.g. COLE etc)**

Questions?

http://t-emu.terma.com/

PoCs:

- Technical: Dr. Mattias Holm <maho@terma.com>
- Sales: Roger M. Patrick <rmp@terma.com>