UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Navigating Exotic SIMD Lands with an LLVM Guide

Marc Solé Bonet, Dr. Leonidas Kosmidis

November 18, 2021

Introduction

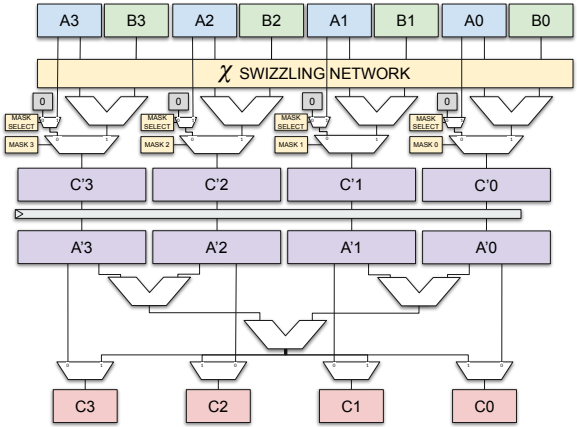
- Increasing interest in artificial intelligence (AI) and machine learning (ML) in space missions: e.g. Mars Perseverance, Φ-Sat-1, OPS-SAT...
- Existing space processors cannot keep up with their computational needs
- Use of COTS devices in institutional missions is challenging:
  - No radiation hardening → cannot be (safely) used beyond LEO
  - Non-space qualified software stacks, lack of RTOS support

SPARROW SIMD Design

- Hardware module implemented for the LEON3 and NOEL-V processors
- Low-cost SIMD design by analyzing the most common ML operations
- Reutilization of the integer register file for reduced overhead
  - Literature shows that 8-bit integers provide small reduction in the inference precision
  - Simplifies the management of the vector data
  - No SIMD load/store instructions are required
- GPU-like features like saturation and swizzling

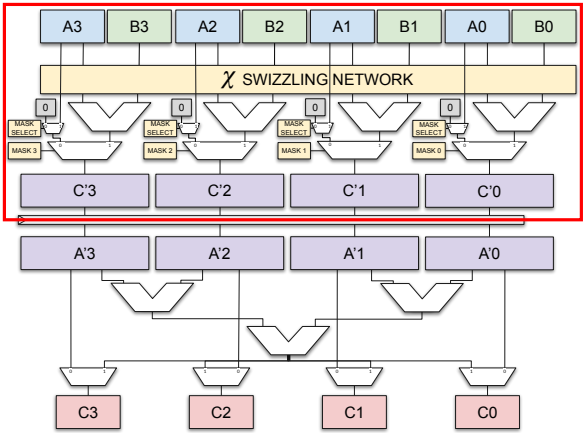
SPARROW Architecture: 4 way 8-bit SIMD, 2 packet VLIW

- First stage: parallel computing
- Second stage: reduction operations
- SPARROW Control Register (SCR)
  - Mask to pass 0s or the original vector
  - Reorder and replication using swizzling network



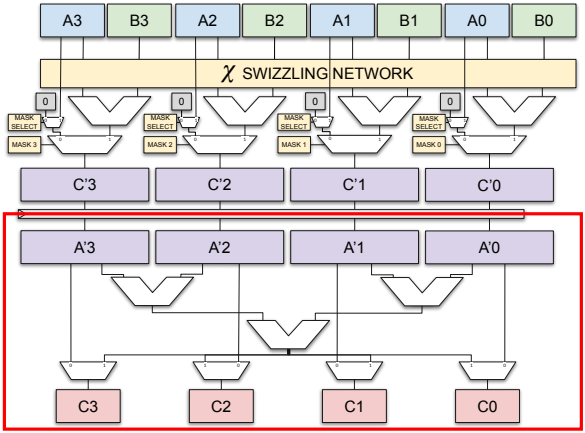
SPARROW Architecture: 4 way 8-bit SIMD, 2 packet VLIW

- First stage: parallel computing
- Second stage: reduction operations
- SPARROW Control Register (SCR)
  - Mask to pass 0s or the original vector
  - Reorder and replication using swizzling network



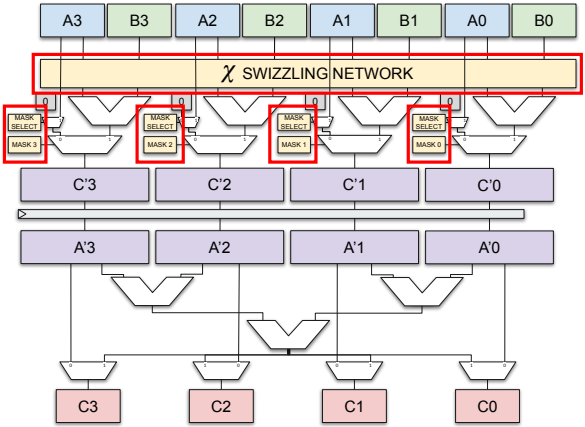
SPARROW Architecture: 4 way 8-bit SIMD, 2 packet VLIW

- First stage: parallel computing
- Second stage: reduction operations
- SPARROW Control Register (SCR)
  - Mask to pass 0s or the original vector
  - Reorder and replication using swizzling network

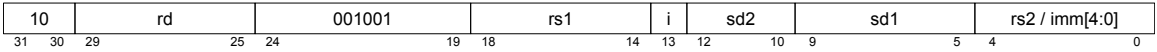


SPARROW Architecture: 4 way 8-bit SIMD, 2 packet VLIW

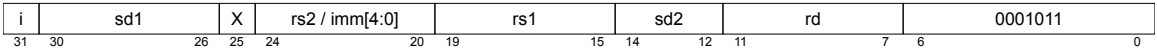
- First stage: parallel computing
- Second stage: reduction operations
- SPARROW Control Register (SCR)
  - Mask to pass 0s or the original vector
  - Reorder and replication using swizzling network



SPARROW instructions



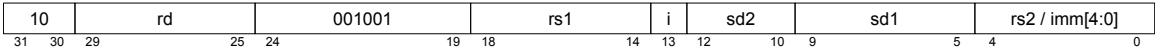
(a) SPARROW SPARC v8 instruction



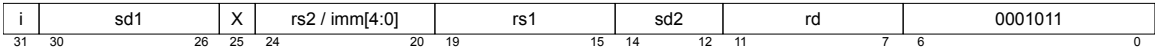
(b) SPARROW RISC-V instruction

- **rd:** Destination register
- **rs1:** First source register
- **rs2:** Second source register
- **imm:** 5-bit encoded immediate
- **i:** Use immediate instead of register for the second operand
- **sd1:** SPARROW first stage operation code
- **sd2:** SPARROW second stage operation code

SPARROW instructions



(a) SPARROW SPARC v8 instruction



(b) SPARROW RISC-V instruction

- **rd:** Destination register
- **rs1:** First source register
- **rs2:** Second source register
- **imm:** 5-bit encoded immediate ➔ frequently used values in ML eg. 0, 1, $2^n(\pm 1)$
- **i:** Use immediate instead of register for the second operand
- **sd1:** SPARROW first stage operation code
- **sd2:** SPARROW second stage operation code

SPARROW Software Support

- No major modifications to the SPARC/RISC-V backends
  - Less than 8h without any prior experience with LLVM
- Only add the new assembly instructions
  - No modification to the IR
  - Allows programming using C inline assembly
    - Small SIMD intrinsics-like library in preprocessor to hide this (not part of this talk)
  - Instructions' names are obtained by combining the operations for both stages with an underscore
  - For certain instructions an alias is available such as `dot` instead of `mul_sum`

Target Programming model

```
unsigned char weights[32*32];
unsigned char next_layer[32*32];
unsigned int a, b, result, scr;

/* set the value of the %scr */
scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
                //swizzling_A = 3-2-2-0,
                //mask_select = 0, mask = 1110

asm("wr %0", %%scr : : "r"(scr));
/* initialise all a components to 0, ie a.xyzw=0 */
a = 0;
/* b.xyzw = weights[0].xyzw */
b = *((unsigned int*) &weights[0]);
asm("nop"); //wait for %scr to commit the write
/* result.xyz = a.xyy + b.zyx */
asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
/* next_layer[0].xyzw = result.xyzw */
*((unsigned int*) &next_layer[0])=result;
```

Target Programming model

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
7                  //swizzling_A = 3-2-2-0,
8                  //mask_select = 0, mask = 1110
9
10 asm("wr %0", %%scr : : "r"(scr));
11 /* initialise all a components to 0, ie a.xyzw=0 */
12 a = 0;
13 /* b.xyzw = weights[0].xyzw */
14 b = *((unsigned int*) &weights[0]);
15 asm("nop"); //wait for %scr to commit the write
16 /* result.xyz = a.xyy + b.zyx */
17 asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
18 /* next_layer[0].xyzw = result.xyzw */
19 *((unsigned int*) &next_layer[0])=result;
```

Target Programming model

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
7                  //swizzling_A = 3-2-2-0,
8                  //mask_select = 0, mask = 1110
9
10 asm("wr %0", %%scr : : "r"(scr));
11 /* initialise all a components to 0, ie a.xyzw=0 */
12 a = 0;
13 /* b.xyzw = weights[0].xyzw */
14 b = *((unsigned int*) &weights[0]);
15 asm("nop"); //wait for %scr to commit the write
16 /* result.xyz = a.xyy + b.zyx */
17 asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
18 /* next_layer[0].xyzw = result.xyzw */
19 *((unsigned int*) &next_layer[0])=result;
```

Target Programming model

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
7                  //swizzling_A = 3-2-2-0,
8                  //mask_select = 0, mask = 1110
9
10 asm("wr %0", %%scr : : "r"(scr));
11 /* initialise all a components to 0, ie a.xyzw=0 */
12 a = 0;
13 /* b.xyzw = weights[0].xyzw */
14 b = *((unsigned int*) &weights[0]);
15 asm("nop"); //wait for %scr to commit the write
16 /* result.xyz = a.xyy + b.zyx */
17 asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
18 /* next_layer[0].xyzw = result.xyzw */
19 *((unsigned int*) &next_layer[0])=result;
```

Target Programming model

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
7                  //swizzling_A = 3-2-2-0,
8                  //mask_select = 0, mask = 1110
9
10 asm("wr %0", %%scr : : "r"(scr));
11 /* initialise all a components to 0, ie a.xyzw=0 */
12 a = 0;
13 /* b.xyzw = weights[0].xyzw */
14 b = *((unsigned int*) &weights[0]);
15 asm("nop"); //wait for %scr to commit the write
16 /* result.xyz = a.xyy + b.zyx */
17 asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
18 /* next_layer[0].xyzw = result.xyzw */
19 *((unsigned int*) &next_layer[0])=result;
```

Backend for SPARROW and Target Machine

- Modify the backend for SPARC (and RISC-V)
- Followed the documentation on *Writing An LLVM Backend* [1]
- We don't require to create a new target `/llvm/lib/Target/Sparc`
- If necessary create a new feature and assign it to a processor
  - In *SPARC.td*

```
def FeatureSPARROW : SubtargetFeature<"sparrow", "IsSPARROW", "true",
                                      "Enable SPARROW instructions">;
```

  - In *SparcSubtarget.h*

```
bool isSPARROW() const { return IsSPARROW; }
```

  - In *SparcInstrInfo.td* or *SparcInstrSPARROW.td*

```
def HasSPARROW : Predicate<"Subtarget->isSPARROW()">
```

[1] LLVM. *Writing an LLVM Backend*. [Visited November 4, 2021]. 2021. URL: https://llvm.org/docs/WritingAnLLVMBackend.html

Register Set

- Reutilization of the register file
- Support for the SCR
  - In *SparcRegisterInfo.td*

    ```
    def SCR : SparcCtrlReg<0, "SCR">;
    ```
  - Add support in *AsmParser/SparcAsmParser.cpp*

Instruction Set

- Use new *SparcInstrSPARROW.td* file to keep the directory organized
- Extended the F3 class from *SparcInstrInfo.td*

```
class F3_AI<bits<5>sd1Val, bits<3> sd2Val, dag outs, dag ins,
            string asmstr, list<dag> pattern, InstrItinClass itin = NoItinerary>
      : F3<outs, ins, asmstr, pattern, itin>{
      bits<5> rs2;

      let op = 2;
      let op3 = 0b001001;

      let Inst{13} = 0;
      let Inst{12-10} = sd2Val;
      let Inst{9-5} = sd1Val;
      let Inst{4-0} = rs2;
}
```

Instruction Set

- Use new *SparcInstrSPARROW.td* file to keep the directory organized
- Extended the F3 class from *SparcInstrInfo.td*

```
class F3_AI<bits<5>sd1Val, bits<3> sd2Val, dag outs, dag ins,
            string asmstr, list<dag> pattern, InstrItinClass itin = NoItinerary>
      : F3<outs, ins, asmstr, pattern, itin>{
      bits<5> rs2;

      let op = 2;
      let op3 = 0b001001;

      let Inst{13} = 0;
      let Inst{12-10} = sd2Val;
      let Inst{9-5} = sd1Val;
      let Inst{4-0} = rs2;
}
```

Instruction Set

- Use new *SparcInstrSPARROW.td* file to keep the directory organized
- Extended the F3 class from *SparcInstrInfo.td*

```
class F3_AI<bits<5>sd1Val, bits<3> sd2Val, dag outs, dag ins,
            string asmstr, list<dag> pattern, InstrItinClass itin = NoItinerary>
      : F3<outs, ins, asmstr, pattern, itin>{
      bits<5> rs2;

      let op = 2;
      let op3 = 0b001001;

      let Inst{13} = 0;
      let Inst{12-10} = sd2Val;
      let Inst{9-5} = sd1Val;
      let Inst{4-0} = rs2
}
```

Instruction Set

- Register and immediate versions

```
multiclass SD_1<string OpcStr, bits<5> sd1Val, bits<3> sd2Val, SDNode OpNode,
                RegisterClass RC, ValueType Ty, Operand immOp,
                InstrItinClass itin = NoItinerary> {

  def rr  : F3_A1<sd1Val, sd2Val, (outs RC:$rd), (ins RC:$rs1, RC:$rs2),
                  !strconcat(OpcStr, " $rs1, $rs2, $rd"),
                  [(set Ty:$rd, (OpNode Ty:$rs1, Ty:$rs2))],
                  itin >;

  def ri  : F3_A1imm<sd1Val, sd2Val, (outs RC:$rd), (ins RC:$rs1, immOp:$simm5),
                  !strconcat(OpcStr, " $rs1, $simm5, $rd"),
                  [(set Ty:$rd, (OpNode Ty:$rs1, (Ty simm5:$simm5)))],
                  itin >;
}
```

Instruction Set

- Register and immediate versions

```
multiclass SD_1<string OpcStr, bits<5> sd1Val, bits<3> sd2Val, SDNode OpNode,
               RegisterClass RC, ValueType Ty, Operand immOp,
               InstrItinClass itin = NoItinerary> {

  def rr  : F3_A1<sd1Val, sd2Val, (outs RC:$rd), (ins RC:$rs1, RC:$rs2),
                  !strconcat(OpcStr, " $rs1, $rs2, $rd"),
                  [(set Ty:$rd, (OpNode Ty:$rs1, Ty:$rs2))],
                  itin >;

  def ri  : F3_A1imm<sd1Val, sd2Val, (outs RC:$rd), (ins RC:$rs1, immOp:$simm5),
                  !strconcat(OpcStr, " $rs1, $simm5, $rd"),
                  [(set Ty:$rd, (OpNode Ty:$rs1, (Ty simm5:$simm5)))],
                  itin >;
}
```

Instruction Set

- Use of `multiclass` to generate all combinations

```
multiclass Simd<string OpcStr, bits<5> sd1Val, SDNode OpNode,
                RegisterClass RC, ValueType Ty, Operand immOp,
                InstrItinClass itin = IIC_iu_instr> {

  defm NOP : SD_1<OpcStr,                          sd1Val, 0b000, OpNode, RC, Ty,
                  immOp, IIC_iu_instr> ;
  defm SUM : SD_1<!strconcat(OpcStr, "sum"),       sd1Val, 0b001, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm MAX : SD_1<!strconcat(OpcStr, "max"),       sd1Val, 0b010, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm MIN : SD_1<!strconcat(OpcStr, "min"),       sd1Val, 0b011, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm XOR : SD_1<!strconcat(OpcStr, "xor"),       sd1Val, 0b100, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
}
```

Instruction Set

- Use of `multiclass` to generate all combinations

```
multiclass Simd<string OpcStr, bits<5> sd1Val, SDNode OpNode,
                RegisterClass RC, ValueType Ty, Operand immOp,
                InstrItinClass itin = IIC_iu_instr> {

  defm NOP : SD_1<OpcStr,                                  sd1Val, 0b000, OpNode, RC, Ty,
                  immOp, IIC_iu_instr> ;
  defm SUM : SD_1<!strconcat(OpcStr, "sum"),               sd1Val, 0b001, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm MAX : SD_1<!strconcat(OpcStr, "max"),               sd1Val, 0b010, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm MIN : SD_1<!strconcat(OpcStr, "min"),               sd1Val, 0b011, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
  defm XOR : SD_1<!strconcat(OpcStr, "xor"),               sd1Val, 0b100, OpNode, RC, Ty,
                  immOp, IIC_iu_simd> ;
}
```

Instruction Set

```
let Predicates = [HasSPARROW], Uses = [SCR] in{
  defm NOP_   : Simd<"nop_",   0b00000, add,  IntRegs, v4i8, simm5Op>;
  defm ADD_   : Simd<"add_",   0b00001, add,  IntRegs, v4i8, imm5Op>;
  defm SUB_   : Simd<"sub_",   0b00010, sub,  IntRegs, v4i8, imm5Op>;
  defm MUL_   : Simd<"mul_",   0b00011, mul,  IntRegs, v4i8, simm5Op>;
  defm MAX_   : Simd<"max_",   0b00101, smax, IntRegs, v4i8, simm5Op>;
  defm MIN_   : Simd<"min_",   0b00110, smin, IntRegs, v4i8, simm5Op>;
  defm AND_   : Simd<"and_",   0b00111, and,  IntRegs, v4i8, imm5Op>;
  defm OR_    : Simd<"or_",    0b01000, or,   IntRegs, v4i8, imm5Op>;
  defm XOR_   : Simd<"xor_",   0b01001, xor,  IntRegs, v4i8, imm5Op>;
}
```

Instruction Set

```
let Predicates = [HasSPARROW], Uses = [SCR] in{
  defm NOP_    : Simd<"nop_",    0b00000, add,  IntRegs, v4i8, simm5Op>;
  defm ADD_    : Simd<"add_",    0b00001, add,  IntRegs, v4i8, imm5Op>;
  defm SUB_    : Simd<"sub_",    0b00010, sub,  IntRegs, v4i8, imm5Op>;
  defm MUL_    : Simd<"mul_",    0b00011, mul,  IntRegs, v4i8, simm5Op>;
  defm MAX_    : Simd<"max_",    0b00101, smax, IntRegs, v4i8, simm5Op>;
  defm MIN_    : Simd<"min_",    0b00110, smin, IntRegs, v4i8, simm5Op>;
  defm AND_    : Simd<"and_",    0b00111, and,  IntRegs, v4i8, imm5Op>;
  defm OR_     : Simd<"or_",     0b01000, or,   IntRegs, v4i8, imm5Op>;
  defm XOR_    : Simd<"xor_",    0b01001, xor,  IntRegs, v4i8, imm5Op>;
}
```

Instruction Set

- Read/Write instructions for the SCR

```
let Predicates = [HasSPARROW], rs2 = 0, rs1 = 0, Uses=[SCR] in
    def RDSCR : F3_1<2, 0b101100,
                (outs IntRegs:$rd), (ins),
                "rd %scr, $rd", []>;

let Predicates = [HasSPARROW], Defs = [SCR], rd=0 in {
    def WRSCRrr : F3_1<2, 0b011001,
                (outs), (ins IntRegs:$rs1, IntRegs:$rs2),
                "wr $rs1, $rs2, %scr", []>;
    def WRSCRri : F3_2<2, 0b011001,
                (outs), (ins IntRegs:$rs1, simm13Op:$simm13),
                "wr $rs1, $simm13, %scr", []>;
}
```

Instruction Set

- Use of alias

```
def : InstAlias<"dot $rs1, $rs2, $rd", (MUL_SUMrr IntRegs:$rd, IntRegs:$rs1,
                                                  IntRegs:$rs2), 0>;
def : InstAlias<"dot $rs1, $simm5, $rd", (MUL_SUMri IntRegs:$rd, IntRegs:$rs1,
                                                    i8imm:$simm5), 0>;
```

Encoding of 5-bit immediates

- Set the encoding method for the 5-bit immediates
- In *SparcInstrSPARROW.td*

```
def simm5Op : Operand<i8> {
  let DecoderMethod = "DecodeSIMM5";
  let EncoderMethod = "getSImm5OpValue";
}
```

- In *Disassembler/SparcDisassembler.cpp* implement `DecodeSIMM5` method
- In *MCTargetDesc/SparcMCCodeEmitter.cpp* implement `getSImm5OpValue` method

Performance

- Similar performance results with handwritten assembly programs for a RISC-V short vector implementation [1]
- CIFAR-10 inference from GPU4S Bench (5.8×) [2,3]
- Similar results with the RISC-V backend

Program	Data size	LEON3	SPARROW	Speedup
Matrix Mult.	256×256	436.788.127	45.016.615	9.70×
Greyscale	256×256	1.054.757	316.973	3.32×
Filter	256×256	11.567.599	3.818.898	3.02×
Polynomial	1024	14.666	3.401	4.31×

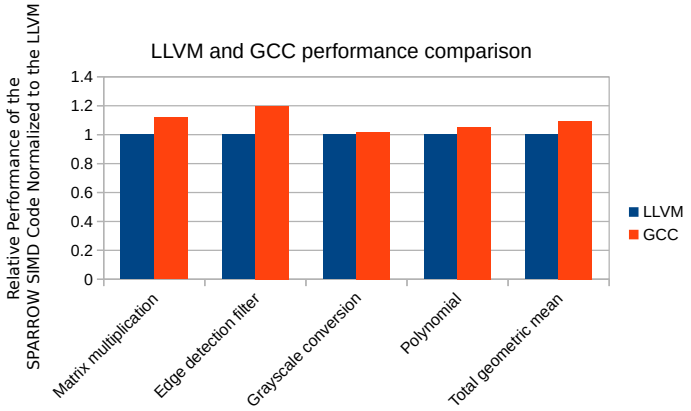
[1] M. Johns and T. J. Kazmierski. "A Minimal RISC-V Vector Processor for Embedded Systems". In: *Forum for specification and Design Languages (FDL)* (2020). DOI: 10.1109/FDL50818.2020.9232940

[2] I. Rodriguez et al. *GPU4S Bench: Design and Implementation of an Open GPU Benchmarking Suite for Space On-board Processing*. Tech. rep. [Visited 11/10/21]. Universitat Politecnica de Catalunya, 2019. URL: https://www.ac.upc.edu/app/research-reports/public/html/research_center_index-CAP-2019,en.html

[3] D. Steenari et al. "OBPMark (On-Board Processing Benchmarks) - Open Source Computational Performance Benchmarks for Space Applications". In: *ESA/DLR/CNES European Workshop on On-Board Data Processing (OBDP)* (2021). URL: https://github.com/OBPMark/GPU4S_Bench

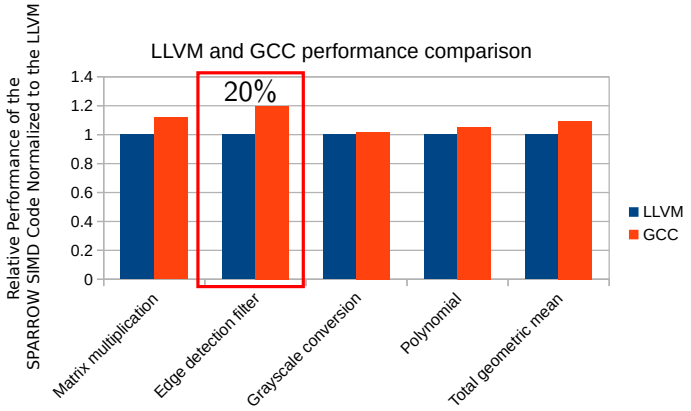
Performance: Comparison with SPARROW Support in GCC

- Relative performance of the SPARROW SIMD code for LLVM and GCC
- GCC 20% faster in the edge detection
- Geometric mean shows GCC is 10% faster overall



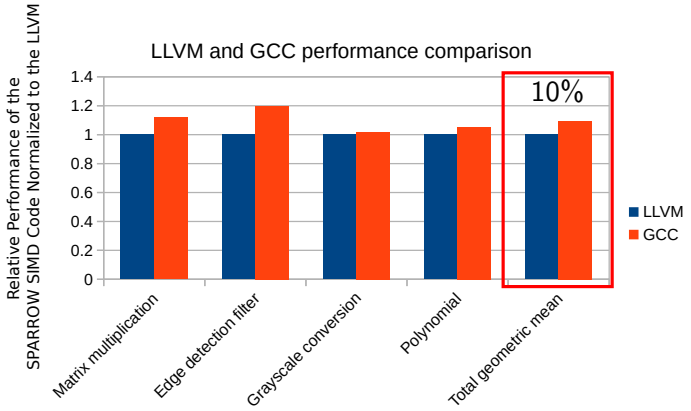
Performance: Comparison with SPARROW Support in GCC

- Relative performance of the SPARROW SIMD code for LLVM and GCC
- GCC 20% faster in the edge detection
- Geometric mean shows GCC is 10% faster overall



Performance: Comparison with SPARROW Support in GCC

- Relative performance of the SPARROW SIMD code for LLVM and GCC
- GCC 20% faster in the edge detection
- Geometric mean shows GCC is 10% faster overall



Vector Extension support

- Clang Language Extensions support vector extensions using `ext_vector_type`[1]
- Allows to use SPARROW 1st stage in C without inline assembly
- Small modifications are required:
  - Include the `v4i8` type in the integer register file in *SparcRegisterInfo.td*
  - Add type compatibility for loads and stores
    - In *SparcInstrSPARROW.td*

```
def : Pat<(v4i8 (load ADDRrr:$addr)), (LDrr ADDRrr:$addr)>;
def : Pat<(store  v4i8:$src, ADDRrr:$addr), (STrr ADDRrr:$addr, $src)>;
```

    - In *SparcISelLowering.cpp*

```
setOperationAction(ISD::STORE, MVT::v4i8, Legal);
setOperationAction(ISD::LOAD, MVT::v4i8, Legal);
```

  - Use `setOperationAction` to set all supported operations to *Legal* or *Expand*
- Current state of the work: Only ADD is supported

[1] LLVM. *Clang Language Extensions*. [Visited November 4, 2021]. 2021. URL: https://clang.llvm.org/docs/LanguageExtensions.html#vectors-and-extended-vectors

Conclusions

- Low-cost SIMD module for AI acceleration
- Portable design in different architectures
- Great speed-up for ML related workloads
  - Over 5× speed-up in complex inference application
  - Over 15× speed-up when using saturation
- Fast and simple implementation of software support in LLVM
  - The process is well documented in *llvm.org/docs*
  - Portable implementation for different LLVM backends (SPARC and RISC-V)
  - General enough for various SIMD architectures, even exotic ones like SPARROW
  - Easy to follow for LLVM beginners

Acknowledgments and Future Work

- This work was partially funded by:
  - ESA under the GPU4S (GPU for Space) project (ITT AO/1-9010/17/NL/AF)
  - The Spanish Ministry of Economy and Competitiveness (MINECO) under grants PID2019-107255GB and FJCI-2017-34095
  - European Commission's Horizon 2020 programme under the UP2DATE project (grant agreement 871465)
  - The HiPEAC Network of Excellence
  - The Xilinx University Program (XUP): **Winner of the Xilinx Open Hardware 2021**

- Future work:
  - Complete Clang vector extension support for 1st and 2nd SIMD stages
  - Extend support using vector-extension to mask and swizzling
  - Auto-vectorization?
  - Support for TVM, MLIR, TensorFlow, PyTorch...

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Navigating Exotic SIMD Lands with an LLVM Guide

Marc Solé Bonet, Dr. Leonidas Kosmidis

November 18, 2021