Compiler support for hardware accelerators
Compiler Innovation

or "What is a compiler support for IMC accelerator"
Grenoble
GdR SOC2024

Henri-Pierre CHARLES

Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France

Thursday, October 3

Introduction : Presentation-Releases

Presentation releases

- V0 MSC Workshop during ESWEEK 2024 : only focus on IMC
- V1 GDR SoC : examples of “support level”
- V2 Fetch 2025

Illustration

Introduction : Amhdal Law's ?

Ahmdal law's: "Speedup is limited by the sequential part"

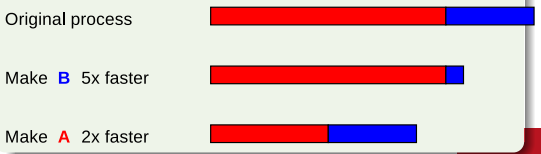
- Acceleration limited by the "X" part : $S = \frac{1}{1-p}$
- X $\in$ {Parallel, Vectorized, using IP bloc, ...}

Programmer approach

- "This part is parallel" let's optimize !
- It's better to fight for a small x2 than for a big x5 !

What to optimize

Two independent parts **A** **B**



Introduction : Niklaus Wirth (15 February 1934, 1 January 2024)

Niklaus Wirth in 2005. Niklaus Wirth

Wirth Projects

- Pascal 1968-1972 Pascal2 / P-Code - UCSD - TurboPascal
- Modula2 1973-76
- Oberon 1977-1980
- Lilith 1977-1981

Books / Articles

- “The Pascal User Manual and Report”
- Algorithms + Data Structures = Programs
- Wirth’s law (1995) **“Software is getting slower more rapidly than hardware is becoming faster.”**
  Article "A Plea for Lean Software"

Introduction : Low Level Programming Models

Programmer view

How to use an IP block from my program ?

How to activate silicon blocks aka IP

1. Independent block
2. CPU Write Control Register / active waiting : device handling in OS
3. Included into ISA + asm intrinsic
4. Include into ISA + IR + code generation

Pro / Cons

1. Nothing to do
2. Huge Programmer Effort : hidden in OS
3. Huge Programmer Effort :
   - cast data in & out
   - manually select instructions
4. Compiler global view :
   - Better global optimization opportunities
   - If IP has possible optimizing parameters -> include higher level optimization
   - automatic correct instruction selection

Compiler Support : Level 0 DMA example

Code https://wiki.st.com/stm32mcu/wiki/Getting_started_with_DMA

```
  /* USER CODE BEGIN 0 */
void XferCpltCallback(DMA_HandleTypeDef *hdma);
uint8_t Buffer_Src[]={0,1,2,3,4,5,6,7,8,9};
uint8_t Buffer_Dest[10];
/* USER CODE BEGIN 0 */

/* USER CODE BEGIN 4 */
void XferCpltCallback(DMA_HandleTypeDef *hdma)
{
  __NOP(); //Line reached only if transfer was successful. Toggle a
}
/* USER CODE END 4 */

/* USER CODE BEGIN 2 */
hdma_memtomem_dma1_Channel1.XferCpltCallback=&XferCpltCallback;
HAL_DMA_Start_IT(&hdma_memtomem_dma1_Channel1,(uint32_t)Buffer_Src
/* USER CODE END 2 */
```

Programming model

- Configuration register setting
- Active wait

Programmer effort

- Want to write `Buffer_Dest = Buffer_Src`
- No language support
- Could be hidden in `memcopy()` but need checks

Compiler action

“nothing”

Compiler Support : Level-1 “ASM Intrinsic”

Code example

https://github.com/borisfoko/Matrix-Multiplication-SIMD-Intrinsics-and-FPU/blob/main/MxM_FPU/MatrixMultiplicationFPU.c

```
// Offset for mat[i][j], w is an row width, t == 1 for transposed
#define mi(w, t, i, j) 4 * ((i * w + j) * (1-t) + (j * w + i) * t)

// Load & multiply.
#define flm(k, i, j, m, n, a, b) \
        __asm fld dword ptr [ebx + mi(m, a, i, k)] \
        __asm fmul dword ptr [ecx + mi(n, b, k, j)]

// Implementation for 6x6 Matrix
#define e6(i, j, l, m, n, a, b) \
        flm(0, i, j, m, n, a, b) \
        flm(1, i, j, m, n, a, b) \
```

Low Level Programming model

- ISA

Programmer effort

- Want to write `M_r = M_a * M_b`
- Need architecture knowledege
- Code specialization depending on the architecture
- Use programming language as a scheduler

Compiler action

- Instruction scheduling
- No semantic level optimization

Compiler Support : Level-2 Language Level

Code example

```
r[i][j] += a[i][k]*b[k][i];
```

Low Level Programming model

- Programming language

Programmer effort

- Want to write `M_r = M_a * M_b`
- Serialize data access

Compiler action

- Low Level Code optimization ++ (need to "recognize" vectors)
- Code selection
- Instruction scheduling
- No object semantic level optimization

Compiler Support : Level-3 Language Level + IR

Code example

```
// a, b and r are matrices
r = a*b
```

Low Level Programming model

- Extended Programming language (DSL)

Programmer effort

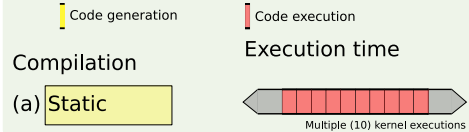
- Minimized

Compiler action

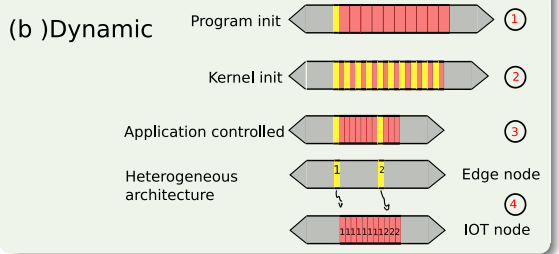
- Specific optimization at object level (arithmetic on objects)
- High level optimization allows to use high level accelerators

Compiler Topics : Execution Scenarios

Static Code Generation

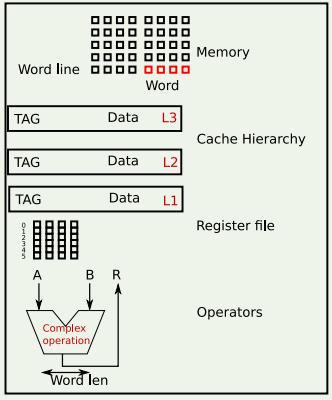


Dynamic Code Generation



Introduction : CISC-versus-RISC

RISC



CISC

"Complex" versus "Reduced" has no meaning.

- RISC : compute instructions, memory instructions
- CISC : compute instructions with memory access (need microcode)

Programming Model : Model-and-Compiler

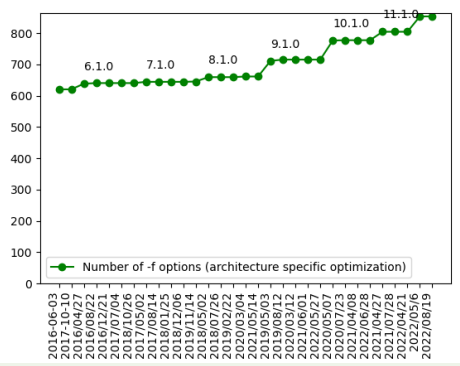
Compiler life (gcc)

- more than 40 year
- more than 100000 files
- precursor in terms of “eco conception”

Compiler Contains

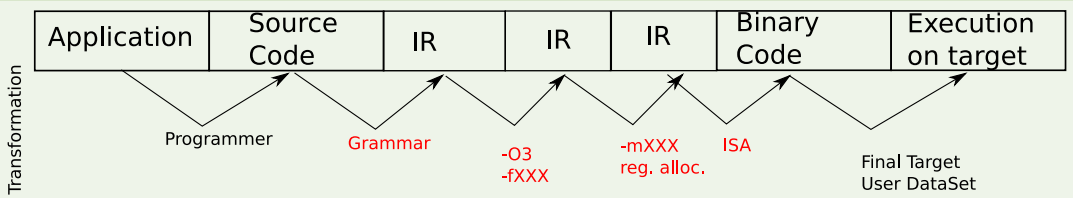
- SSA form : program as transformable data. Program transformation : parallelization, vectorization ...
- Register allocation.
- Instruction scheduling : based on data type arithmetics
- Asumptions about target
- Pattern matching for low level instructions selection

Illustration



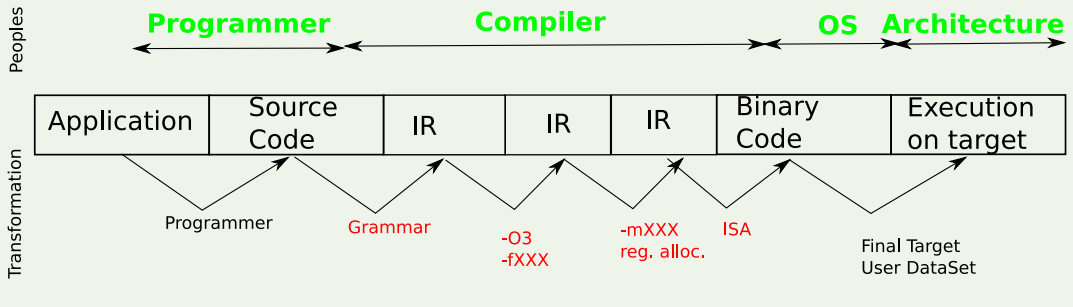
Compiler Support : Compiler And Architecture Links

Links between application and architecture



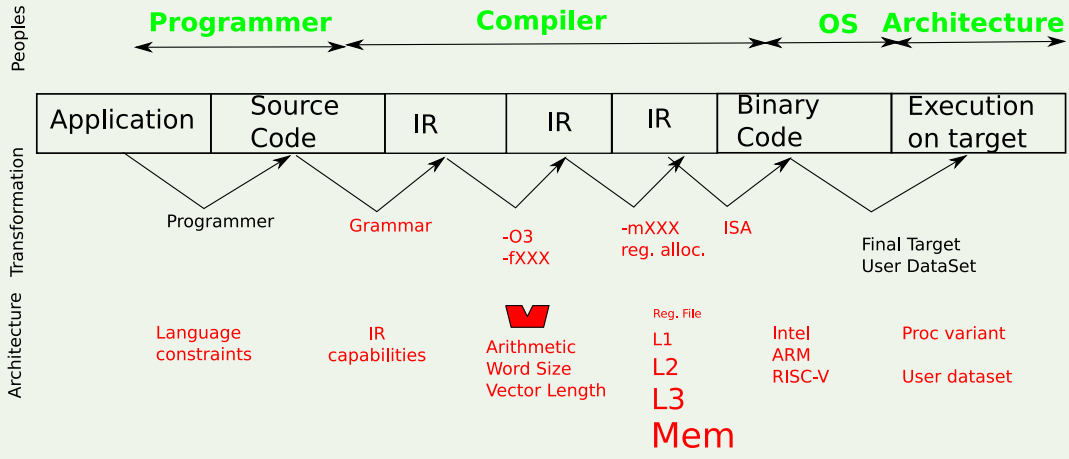
Compiler Support : Compiler And Architecture Links

Links between application and architecture



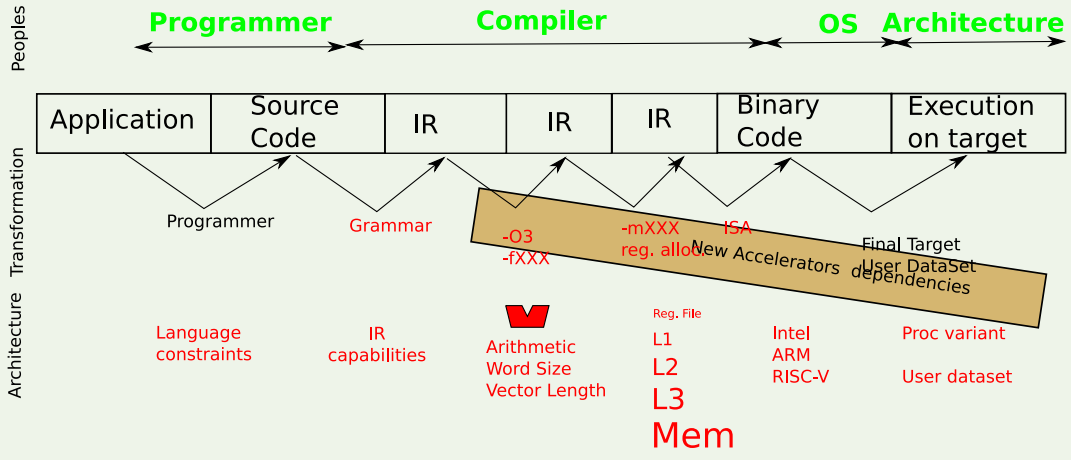
Compiler Support : Compiler And Architecture Links

Links between application and architecture



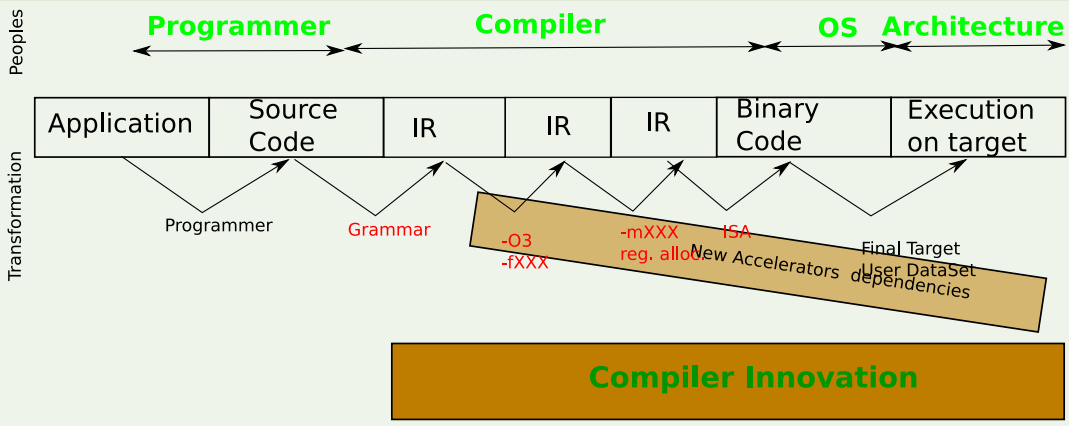
Compiler Support : Compiler And Architecture Links

Links between application and architecture



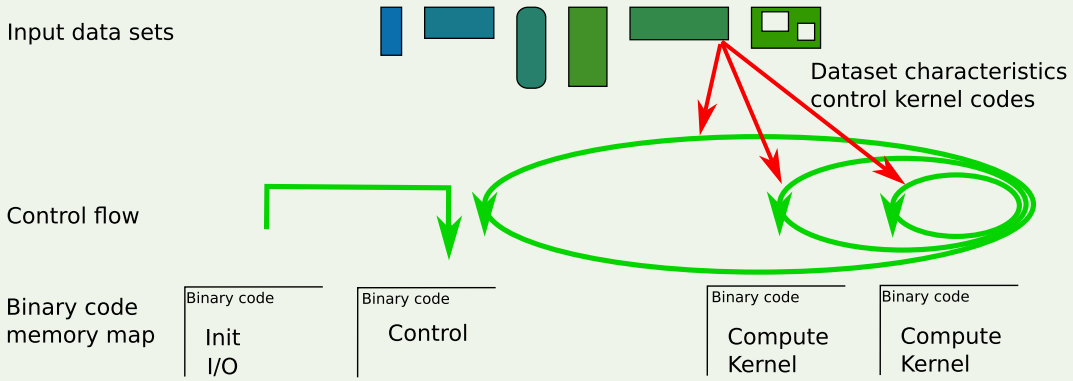
Compiler Support : Compiler And Architecture Links

Links between application and architecture



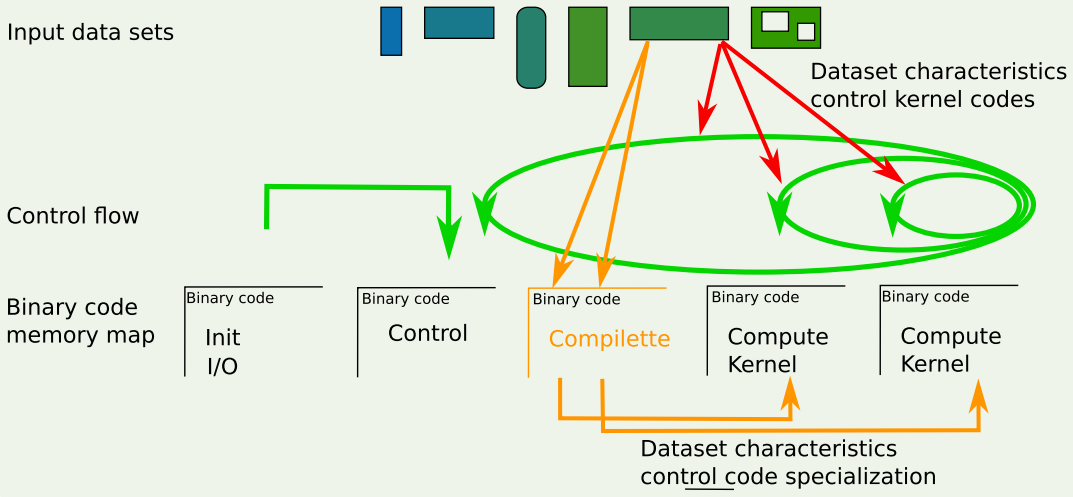
Compilette principle : “Working Example”

Control flow in application



Compilette principle : "Working Example"

Control flow in application with dynamic adaptation



List of Code Generation Scenarios

Compilation scenarios

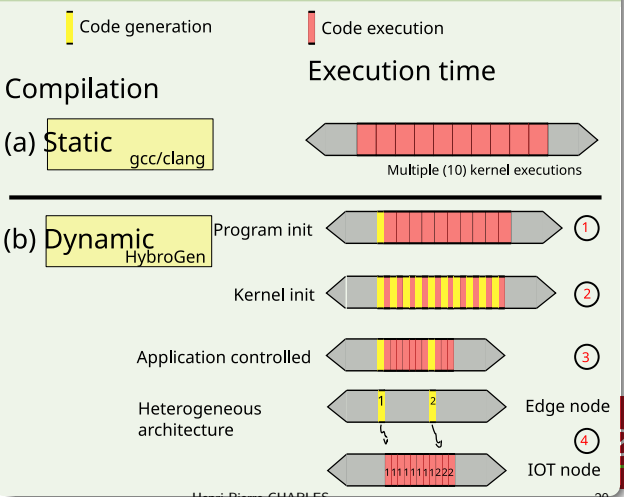
- (a) Static compilation
- (b) Dynamic adaptation
  1. Program initalization
  2. Kernel initialization
  3. Application controlled
  4. Heterogeneous architecture (multi-isa support)

Target list

- RISCV (Embedded system)
- CSRAM (Embedded system)
- POWER 8 (HPC computer)
- AARCH64 (both)
- Others (both)

All following scenarios examples works on all platforms

Illustration



Compiler Support : HybroLang DSL description

Specific features

- C like syntax
- Variable are hardware registers
- Mix run time data values and binary code
  - `#( C expression )` include C expression
- Datype triplet
  `arithmetic wordlen vectorlen`
  - `int 32 1` scalar int
  - `flt 32 2` vector of 2 floats
  - `flt 32 #(vlen) vlen` vector of floats
  - `flt #(wlen) 4` vector of 4 floats of size `wlen`

"Multi-time" Code Generation

- Static time : Generate binary code generator
  - Included into compilation chain, remplace a part of the C code
- Run-time : Generate binary code
  - Faster than any JIT
  - Small code generator able to fit on embedded platforms

HybroGen : Simple-Add-Source

Simple Addition with specialization

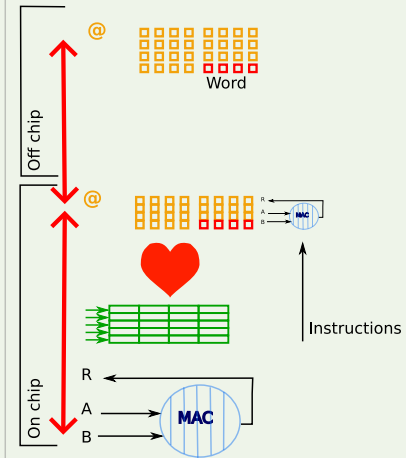
```
typedef  int (*pifi)(int);

pifi genAdd(pifi ptr, int  b)
{
  #[
  int 32 1 add (int 32 1 a)
  {
    int 32 1 r;
    // b values will be included in code generation
    r =  a + #(b);
    return r;
  }
  ]#
  return (pifi) ptr;
}
```

Compilette usage

```
// Generate instructions
fPtr = genAdd (fPtr, in0);
// Call generated code
res  = fPtr(in1);
```

Examples : CSRAM (Computational SRAM)

Architecture



Programmer view

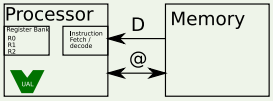
- Single program flow
- Non Von Neumann model : CPU send instructions to CSRAM
- DSL approach, which express
  - Heterogeneous computation (DONE)
  - Memory hierarchy (ONGOING)

Software support

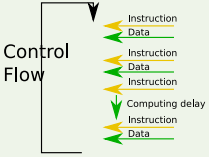
- HybroLang compiler `https://github.com/CEA-LIST/HybroGen`
- Functionnal emulator (based on QEMU): `https://github.com/CEA-LIST/csram-qemu-plugin`

Inverted Von Neumann Programming Model

Choosen Programming model



Bus transactions Code

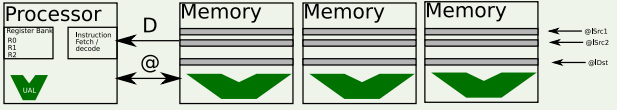


ld r0 = @A

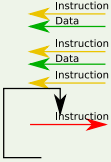
ld r1 = @B

add r2 = r0, r1

st r2 = @C



Bus transactions Code



ld r0 = @lSrc1

ld r1 = @lSrc2

ld r2 = @lDst

add lDst = lSrc1, lSrc2

increment line adresses

Why ?

- Allows scalability :
  - Any vector size
  - Any tile number
  - Any system configuration : near or far IMC
- Works with any processor

Programming Model : Image Diff

Mini code Example : HybroLang code example

```
pifiii genSubImages(h2_insn_t * ptr){
#[
  int 32 1 subImage(int[] 16 8 a, int[] 16 8 b, int[] 16 8 res, int 32 1 len)
    {
      int 32 1 i; // int 32 1 = RISC-V register
      // int[] 16 8 = array of C-SRAM lines
      for (i = 0; i < len; i = i + 1) // Control done on RISC-V
        {
          res[i] = a[i] - b[i];       // Workload done on C-SRAM
        }
    }
  return 0;
]#
  return (pifiii) ptr;}
```

Compiler support

- Dynamic interleaving
- Instruction generator generator notion

CSRAM : Compiler Support

How to generate code for CxRAM

- Recognize operation on `int X Y` where X = 128
- Replace this operation by sequence of instruction
  - Set Rs1 and Rs2 with dest, src1, src2 memory line
  - store instruction
  - 1 bit which say “this is not a store”
  - register parameters

Instruction type

Logic and integer arithmetic (8, 16, 32 bits)

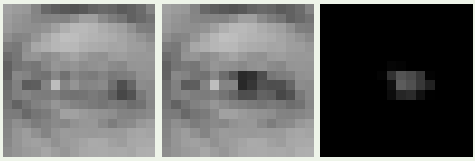
- R-type $M_{dest} = M_{src1} X M_{src2}$
- I-type $M_{dest} = M_{src1} X Const$
- U-type $M_{dest} = Const$

HybroGen : ImageDiff-Run

CxRAM Usage

- Compute image difference
- Iterate on image lines (RISCV)
- Use difference operators / 16 pixels wide (CxRAM)

Dataset



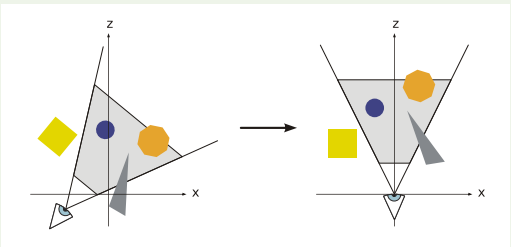
Experimentation : Graphics Pipeline

Graphics_pipeline

- Application -> Geometry -> Rasterization -> Screen
- "World coordinate system" based on 4x4 matrices
- Who is in charge of the eye position ?

The World Coordinate System Homogeneous coordinates

- Translation : $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$
- Rotation around Z axis: $\begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$



Optimization opportunities

0 remove 2 operations

1 remove 1 operation

power of 2 replace mul by shift

Experimentation : Matrix x Vector “legacy”

MESA library implementation

https://gitlab.freedesktop.org/mesa/mesa/-/blob/main/src/mesa/math/m_matrix.c
Is this implementation strange ? How many flops ?

```
void
_mesa_transform_vector( GLfloat u[4], const GLfloat v[4], const GLfloat m[16] )
{
   const GLfloat v0 = v[0], v1 = v[1], v2 = v[2], v3 = v[3];
#define M(row,col)  m[row + col*4]
   u[0] = v0 * M(0,0) + v1 * M(1,0) + v2 * M(2,0) + v3 * M(3,0);
   u[1] = v0 * M(0,1) + v1 * M(1,1) + v2 * M(2,1) + v3 * M(3,1);
   u[2] = v0 * M(0,2) + v1 * M(1,2) + v2 * M(2,2) + v3 * M(3,2);
   u[3] = v0 * M(0,3) + v1 * M(1,3) + v2 * M(2,3) + v3 * M(3,3);
#undef M
}
```

Experimentation : MxV-compilette

Compilette code anatomy

```
typedef int d_t;
typedef d_t matrix_t [T][T];
typedef d_t vector_t [T];
typedef d_t (*pifvv)(vector_t, vector_t);

pifvv genVectorMatrixProduct(pifvv ptr, matrix_t m)
{
  #[
  int 32 1 add (int[] 32 1 o, int[] 32 1 i)
   {
     o[0] = i[0]*#(m[0][0]) + i[1]*#(m[1][0]) + i[2]*#(m[2][0]) + i[3]*#(m[3][0]);
     o[1] = i[0]*#(m[0][1]) + i[1]*#(m[1][1]) + i[2]*#(m[2][1]) + i[3]*#(m[3][1]);
     o[2] = i[0]*#(m[0][2]) + i[1]*#(m[1][2]) + i[2]*#(m[2][2]) + i[3]*#(m[3][2]);
     o[3] = i[0]*#(m[0][3]) + i[1]*#(m[1][3]) + i[2]*#(m[2][3]) + i[3]*#(m[3][3]);
   }
  return o[3];
  ]#
  return ptr;
}
```

Experimentation : MxV Generated Code for "Dense" matrix

Generated code (under debugger)

```
0x420790: ldr w2, [x1]
0x420794: lsl w6, w2, #2
0x420798: add x7, x1, #0x4
0x42079c: ldr w3, [x7]
0x4207a0: mov w9, #0x5        // #5
0x4207a4: mul w10, w3, w9
0x4207a8: add x11, x6, x10
0x4207ac: add x6, x1, #0x8
0x4207b0: ldr w4, [x6]
0x4207b4: mov w10, #0x9       // #9
0x4207b8: mul w12, w4, w10
0x4207bc: add x13, x11, x12
0x4207c0: add x11, x1, #0xc
0x4207c4: ldr w5, [x11]
0x4207c8: mov w12, #0xd       // #13
0x4207cc: mul w14, w5, w12
0x4207d0: add x15, x13, x14
0x4207d4: str w15, [x0], #0
0x4207d8: add x6, x0, #0x4
```

Result

Input matrix $\begin{bmatrix} 4 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$

- 54 insn generated
- Should be similar to the legacy code
- Without matrix load !
- Optimization on specific values (power of 2)

Experimentation : MxV-Generated Code Unity Matrix

Argumentation

```
0x420790: ldr w2, [x1]
0x420794: add x6, x1, #0x4
0x420798: ldr w3, [x6]
0x42079c: add x7, x1, #0x8
0x4207a0: ldr w4, [x7]
0x4207a4: add x9, x1, #0xc
0x4207a8: ldr w5, [x9]
0x4207ac: str w2, [x0], #0
0x4207b0: add x6, x0, #0x4
0x4207b4: str w3, [x6], #0
0x4207b8: add x6, x0, #0x8
0x4207bc: str w4, [x6], #0
0x4207c0: add x6, x0, #0xc
0x4207c4: str w5, [x6], #0
0x4207c8: ret
0x4207cc: udf #0
```

Illustration

Input matrix Id $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

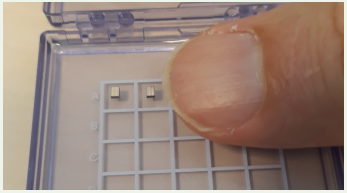
- 15 insn generated
- Huge optimization

CxRAM-Status : Circuit

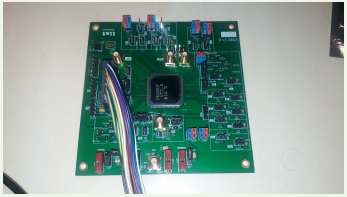
Chip design evolution

- 1 chip built, characterized : CSRAM part only, (photo)
- Result published : "A 35.6TOPS/W/mm² 3-Stage Pipelined Computational SRAM with Adjustable Form Factor for Highly Data-Centric Applications" 2020
- 1 chip built, under testing / characterization : CSRAM + RISCV
- Ongoing work on new instruction set variants

IMPACT circuit (2019)



RISCV and CSRAM under testing (2023)



Conclusion : Conclusion

Architecture point of view

- Application ? Future is not only based on deep learning !
- Parallelism type
- **Memory layout is a key** !
  - DRAM interleaving
  - Data locallity / aligment

Tools for collaborations

- DSL / Compiler : `https://github.com/CEA-LIST/HybroGen`
- Emulator, based on QEMU : `https://github.com/CEA-LIST/csram-qemu-plugin`

HybroGen Roadmap

- Include data value based run-time optimization (Already started)
- Include explicit data movement for sparse accelerators
- Include variable precision floating point number
- Include system level PIM capabilities
- more to come on the road