Quels choix possibles dans les modèles d'exécution bas niveau ?

Allo Houston ? On a un problème, j'ai perdu la mémoire

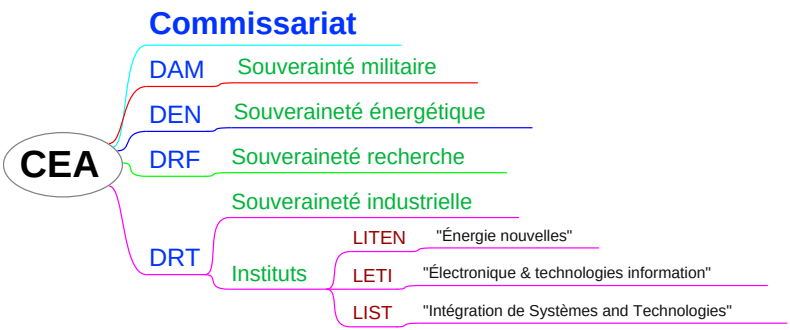
Réunion pousse café / DSCIN

Henri-Pierre CHARLES

CEA DSCIN department / Grenoble

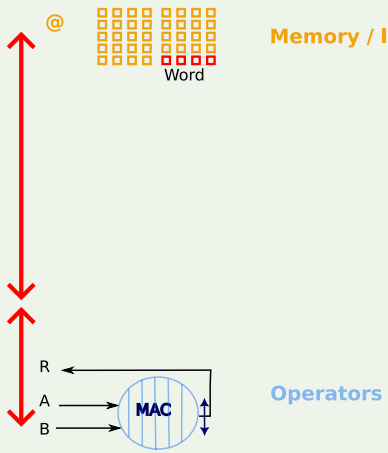
Mon 03 Apr 2023

Introduction : Présentation-CEA



Components : Von-Neumann

Von Neumann Model



Compiler view

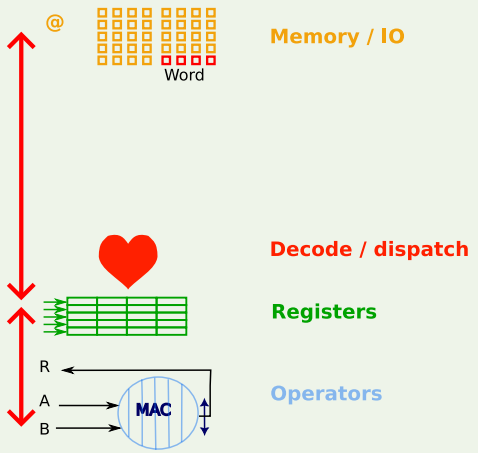
- 0x0000800000004970 += 0x0000800000004974 * 0x0000800000004978

Programmer view

- R += A * B

Components : Von-Neumann (registers as L0)

Von Neumann Model



Instruction

Ops number

Regs number

Op	Reg D	Reg S1	Reg S2

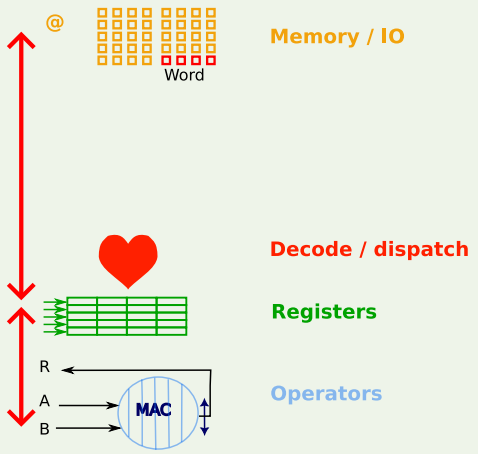
32 bits

Algorithme

```
Toujours faire :
    Fetch insn at address PC (IF)
    Increment PC
    Decode insn (DE)
    Execute insn (EXE)
    Store result (WB)
```

Components : Von-Neumann (registers as L0 & instructions)

Von Neumann Model



Compiler view

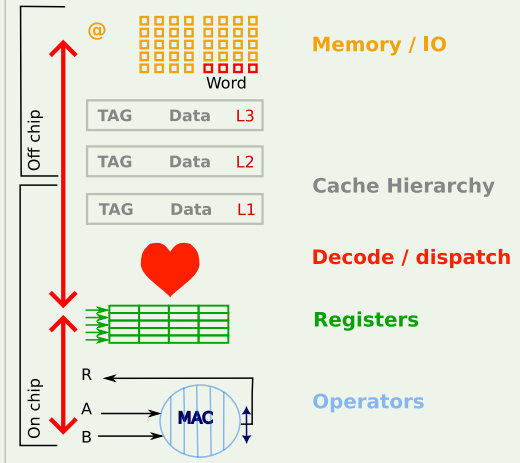
- ld r0, 0x0000800000004974
- ld r1, 0x0000800000004978
- mac r3, r0, r1
- st r3, 0x0000800000004970

Programmer view

- R += A * B

Components : Von-Neumann + cache hierarchy

Von Neumann Model



Compiler view

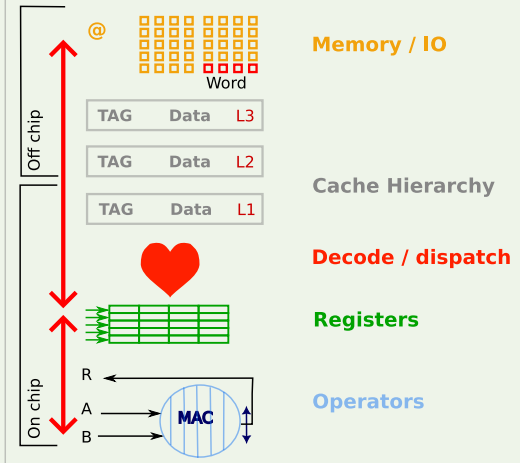
- What is SIZE value ?
- What is `sizeof (A[0][0])` ?
- What are cache size ?
- Data aligments ?
- What if Line# != Column# ?
- What is `typeof(A[0][0])`

Programmer view

```
for (int l = 0; l < SIZE; l++)
  for (int c = 0; c < SIZE; c++)
    for (int k = 0; k < SIZE; k++)
      R[l][c] += A[l][k] * B[k][c];
```

Components : Von-Neumann + cache hierarchy

Von Neumann Model



Compiler view

- **Because compiler fail at portable performance**

Programmer view

```
for (c= 0; c<NCOL; c+=cacheLineSize)
 for (l= 0; l<NLINE; l+=halfCacheLine)
  for (c2= 0; c2<NCOL; c2+=halfCacheLine)
    for (lk= 0; lk<halfCacheLine; lk++)
     for (c2k= 0; c2k<halfCacheLine; c2k++)
      for (ck= 0; ck<cacheLineSize; ck++)
        res[l+lk][c2+c2k]+=      a[l+lk] [c+ck]* b[c2+c2k][c+ck];
```

Learn to program = learn to serialize / schedule on defined hardware !

GPU / CUDA

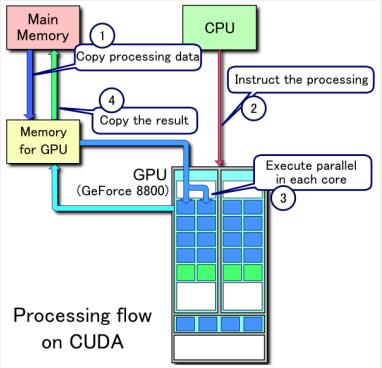
Characteritics

- Programming language for data parallelism
- CUDA

Illustration

- Normalized: No
- Portable : No (NVIDIA only)
- Scalable : No, why ?
- Asynchronous : no way to schedule
- At least 2 instruction flow
- Data dependent essential to performance (blocking)
- Need hardware thread support
- Specialized ISA (PTX)

Example of CUDA processing flow



GPU CUDA-Example

Header and CUDA code

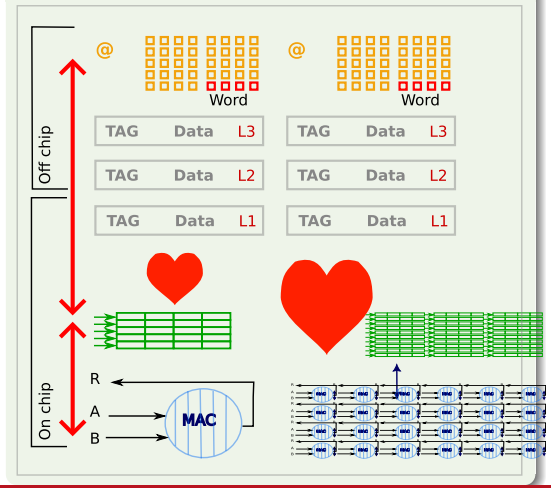
```
import pycuda.compiler as comp
import pycuda.driver as drv
import numpy
import pycuda.autoinit

mod = comp.SourceModule("""
__global__ void multiply_them(float *dest, float
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")
```

Python code

```
multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(400,1,1))

print dest-a*b
```

GPU : Example

CPU / GPU



How to handle code heterogeneity

Programmer view (C / C++ / Fortran / Python)

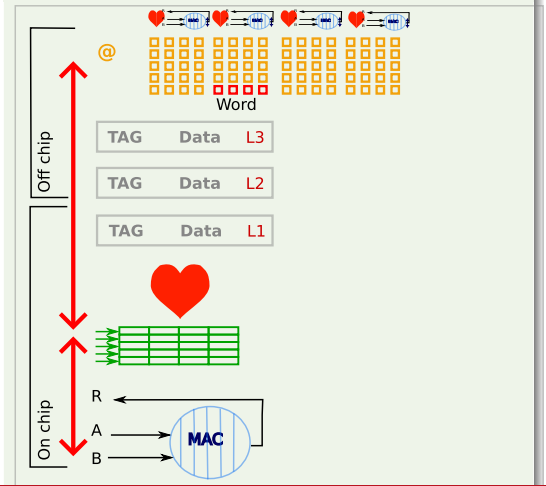
- IO
- Memory tiling / copy
- GPU scheduling
- Synchronization

Compiler view (CUDA)

- Local view
- Hardware thread support
- “Local” computation
- “Register allocation”

Examples : UPMEM

CPU / GPU



Characteristics

- Processing in DRAM
- Multipe instruction flows
- `https://github.com/upmem`

Programmer view

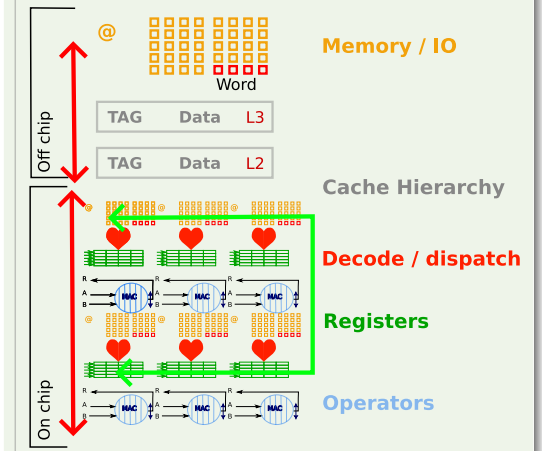
- OpenMP programming
- Scalar computation
- Should use hardware thread support

Compiler view

- Use specialized ISA
- No global view

Examples : Kalray

CPU / GPU



Programmer view (C / C++ / Fortran / Python)

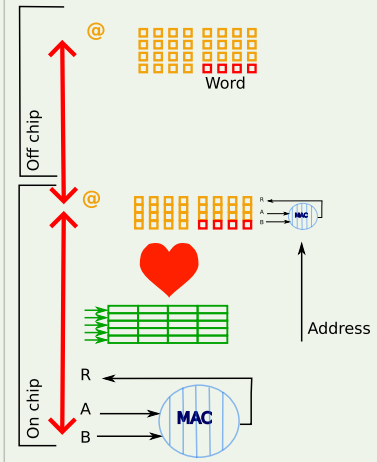
- IO
- Memory tiling / copy
- GPU scheduling
- Synchronization
- Possible local communication

Compiler view (C)

- Local view
- Explicit memory management (scratchpad memory)
- “Local” computation

Examples : Michigan

Compute cache



Compiler view

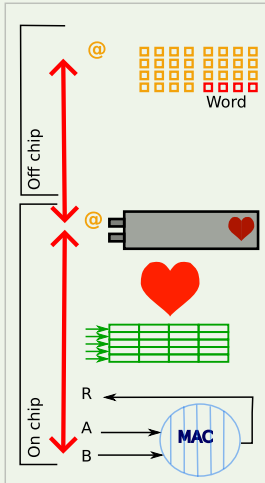
- Nop

Programmer view

- Has to micro program:
  - Data locality
  - Get data = trigger computation
- Use specialized addresses to activate computation
- Bit serial computation (slow)
- Large potential parallelism (?)

Examples : Accelerator

Hardware Accelerator



Programmer view

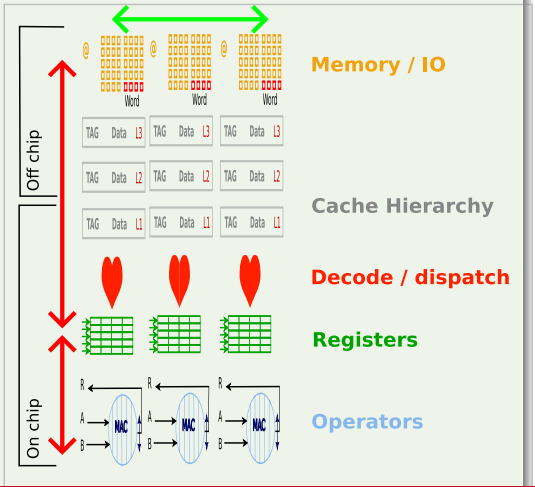
- Write in configure register (memory mapped)
- Synchronize
- “Black box” accelerator
- Good acceleration for a specialized / complex function
- Not included in the instruction flow. I.e. not included in compiler optimization.

Compiler wiew

- Nop
- Can not optimize / specialize

Example: MPI (Message Passing Interface)

CPU only



Programmer view (C / C++ / Fortran / Python)

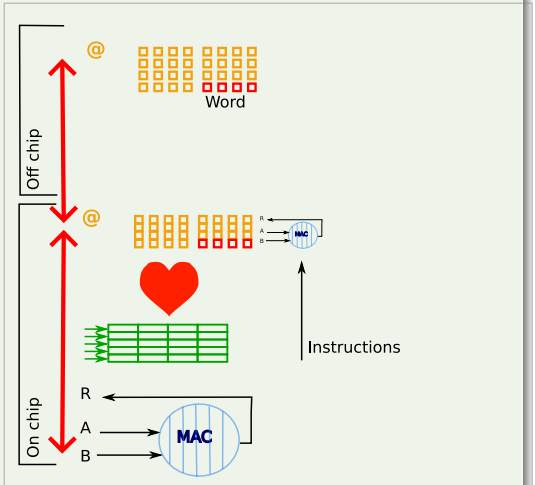
- Local view ()
- "Local" computation
- Data explicit management, global communication
- Excellent possible scalability, up to 10^6 nodes
- "Possible" but not free

Compiler view (C, Fortran)

- Local code generation
- Could be different
- Are similar for practical reasons

Examples : CSRAM (Computational SRAM)

Architecture



Programmer view

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

Compiler

- `https://github.com/CEA-LIST/HybroGen`
- Functionnal emulator (based on QEMU):

Intro : CSRAM Memory

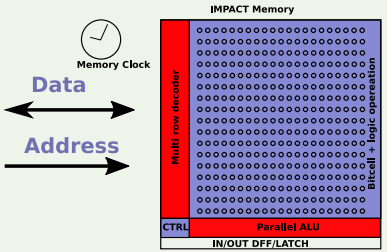
Memory with

- Bitcell 2 ports
- Vector like alu
- Multi-line selector
- No internal instruction execution

Design choices

- Minimize CPU modification
- Only one instruction sequence (CPU master)

IMPACT memory



Technology

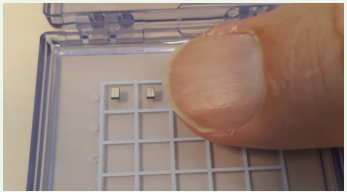
SRAM technology (FDSOI 22nm)

CxRAM-Status : Circuit

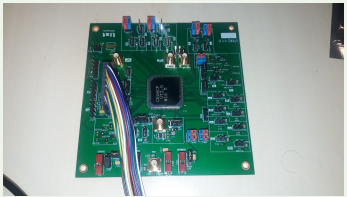
Chip design evolution

- 1 chip built, characterized : CSRAM part only, (photo)
- Result published : "A 35.6TOPS/W/mm^2 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

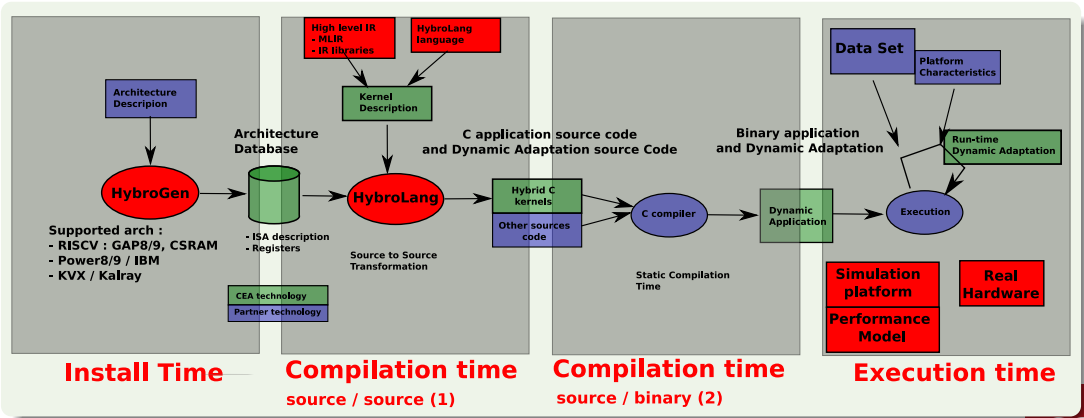
IMPACT circuit (2019)



RISCV and CSRAM under testing (2023)



HybroGen: General View



Compiler : Compilation Flow

HybroLang compilation flow

HybroGen
- Lex / Parser (antlr)
- IR
- Code generator generator

Binary code generation any C compiler

Execution dynamic binary code generation

QEMU (+ plugin)
Silicon + OS
Bare metal silicon

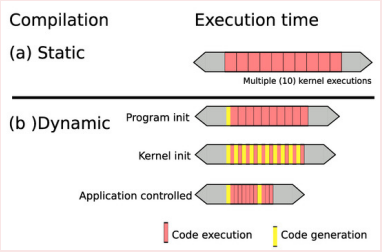
Why dynamic adaptation

- Speed
- Security
- Heterogeneity

Compilation flow for CXRAM

- Specific optimization
- Generate parameter in registers
- Generate RISCV store word

Compilation scenarios



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;
          for (i = 0; i < len; i = i + 1)
                {
                  res[i] = a[i] - b[i];
                }
        }
  return 0;
]#
  return (pifiii) ptr;
}
```

Compiler support

- Dynamic interleaving

HybroGen : ImageDiff-Generated Compilette 1/2

Compilette

```
pifiii genSubImages(h2_insn_t * ptr)
{
/* Code Generation of 51 instructions */
/* Symbol table :*/
        /*VarName = { ValOrLen, arith, vectorLen, wordLen, regNo, Value} */
        h2_sValue_t a = {REGISTER, 'i', 1, 32, 10, 0};
        h2_sValue_t b = {REGISTER, 'i', 1, 32, 11, 0};
        h2_sValue_t res = {REGISTER, 'i', 1, 32, 12, 0};
        h2_sValue_t len = {REGISTER, 'i', 1, 32, 13, 0};
        h2_sValue_t h2_outputVarName = {REGISTER, 'i', 1, 32, 10, 0};
        h2_sValue_t i = {REGISTER, 'i', 1, 32, 5, 0};
        h2_sValue_t h2_00000009 = {REGISTER, 'i', 1, 32, 6, 0};
        h2_sValue_t h2_00000012 = {REGISTER, 'i', 1, 32, 6, 0};
```

HybroGen : ImageDiff-Generated-Main

Main program

```
riscv_genLABEL(h2_prologue_00000000);
riscv_genMV_2(i, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, 0});
riscv_genLABEL(h2_begin_00000001);
riscv_genBGE_3(i, len, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, (int)((labelAddr
riscv_genLUI_2(h2_00000046, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, 532032});
riscv_genADD_3(h2_00000050, res, i);
riscv_genMV_2(h2_00000051, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, 2});
riscv_genSL_3(h2_00000051, h2_00000050, h2_00000051);
riscv_genOR_3(h2_00000051, h2_00000046, h2_00000051);
riscv_genADD_3(h2_00000055, a, i);
riscv_genADD_3(h2_00000059, b, i);
riscv_genMV_2(h2_00000060, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, 16});
riscv_genSL_3(h2_00000060, h2_00000059, h2_00000060);
riscv_genOR_3(h2_00000060, h2_00000055, h2_00000060);
riscv_genW_3(h2_00000051, h2_00000060, (h2_sValue_t) {VALUE, 'i', 1, 32, 0,
riscv_genMV_2(h2_00000065, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, 1});
riscv_genADD_3(i, i, h2_00000065);
riscv_genBA_2((h2_sValue_t) {REGISTER, 'i', 1, 32, 0, 0}, (h2_sValue_t) {VAL
riscv_genLABEL(h2_end_00000002);
riscv_genRET_0();
```

HybroGen : ImageDiff Main

Main program

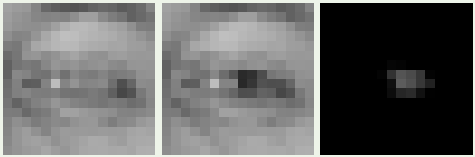
```
readImage (argv[1], in1);
readImage (argv[2], in2);
ptr = h2_malloc (1024);
code = genSubImages(ptr); // Code generation
code (in1, in2, res, 32); // Code call
```

HybroGen : ImageDiff-Run

CxRAM Usage

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

Dataset



HybroGen : ImageDiff-Build

Build and run an application

```
RunDemo.py -a cxram -i CxRAM-ImageDiff
Namespace(arch=['cxram'], clean=False, debug=False, inputfile=['CxRAM-ImageDiff'], v
-->rm -f CxRAM-ImageDiff CxRAM-ImageDiff.c
-->which riscv32-unknown-linux-gcc
-->../HybroLang.py --toC --arch cxram --inputfile CxRAM-ImageDiff.hl
-->riscv32-unknown-linux-gcc -o CxRAM-ImageDiff CxRAM-ImageDiff.c
('MonOeilGrisFerme.pgm', 'MonOeilGris.pgm')
-->qemu-riscv32 CxRAM-ImageDiff MonOeilGrisFerme.pgm MonOeilGris.pgm
```

HybroGen : ImageDiff Instruction Selector

Main program

```
void riscv_genSL_3(h2_sValue_t P0, h2_sValue_t P1, h2_sValue_t P2)
{

    if ((P0.arith == 'i') && (P0.wLen <= 32) && (P0.vLen == 1) && P0.ValOrReg == REG
    {
        RV32I_SLLI_RRI_I_32_1(P0.regNro, P1.regNro, P2.valueImm);
    }

    else if ((P0.arith == 'i') && (P0.wLen <= 32) && (P0.vLen == 1) && P0.ValOrReg =
    {
        RV32I_SLL_RRR_I_32_1(P0.regNro, P1.regNro, P2.regNro);
    }

    else
    {
        printf("Warning, generation of SL is not possible with this arguments P0 : %c
        h2_codeGenerationOK = false;
    }
}
```

HybroGen : ImageDiff : Generated Macros

Main program

```
#define riscv_G32(INSN){ *(h2_asm_pc++) = (INSN);}

#define RV32I_RET__I_32_1() /* RET */ \
do { \
        riscv_G32(((0x8067 >> 0) & 0xffffffff)); \
} while(0)

#define RV32I_SLLI_RRI_I_32_1(r3,r1,i0) /* SL */ \
do { \
        riscv_G32(((0x0 & 0x7f) << 25)|((i0 & 0x1f) << 20)|((r1 & 0x1f) << 15)|((0x1
} while(0)

#define RV32I_SLL_RRR_I_32_1(r3,r1,r2) /* SL */ \
do { \
        riscv_G32(((0x0 & 0x7f) << 25)|((r2 & 0x1f) << 20)|((r1 & 0x1f) << 15)|((0x1
} while(0)

#define RV32I_ADDI_RRI_I_32_1(r1,r0,i0) /* ADD */ \
```

CxRAM-Status : Software Tools

Code generation

- HybroGen compiler first public release `https://github.com/CEA-LIST/HybroGen`
- Emulator, based on QEMU : not (yet) open source
- Full development chain :
  - Cross compiler (gcc)
  - Cross source level debugger (gdb)
- 5 CsRAM instruction set variants support

High level Applications

- OpenCV support (image filtering)
- Tensor flow lite
- Working on :
  - Cryptography

Conclusion : Conclusion

Architecture point of view

- Application ? Futur is not only based on deep learning !
- Parallelism type
- **Memory layout is a key** !

Programmer view

- Only one code source
- Keep it simple
- Provide hardware information

Compiler view

- How to deal with heterogenity
- Inference on Code source
- **Compilation for Low level architecture is a key**