In-memory Computing Low Level Programming Model & Compiler Innovation

or "What is a compiler support for IMC accelerator"
Pearls on Tool Chains of In-memory Computing
MSC@ESWEEK 2024

Henri-Pierre CHARLES

CEA DSCIN department / Grenoble

Thursday, October 3

Introduction : Amhdal Law's ?

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

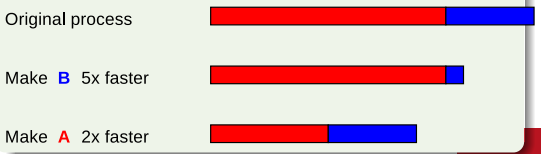
- Acceleration limited by the "X" part : $S = \frac{1}{1-p}$
- X $\in$ {Parallel, Vectorized, using IP bloc, ...}

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

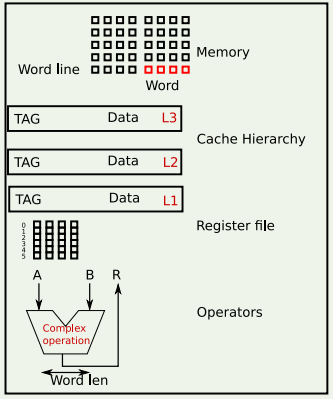
1. Independent block
2. CPU Write Control Register / active waiting : device handling in OS
3. Included into ISA + asm intrinsic
4. Include into ISA + IR + code generation

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

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

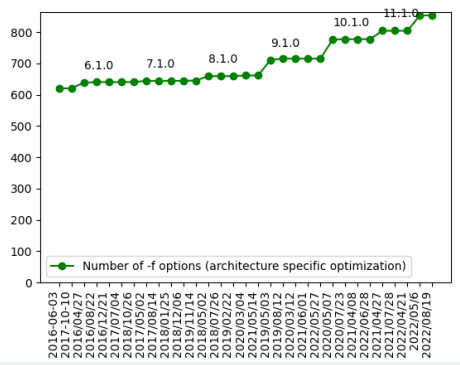
Compiler life (gcc)

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

Compiler Contains

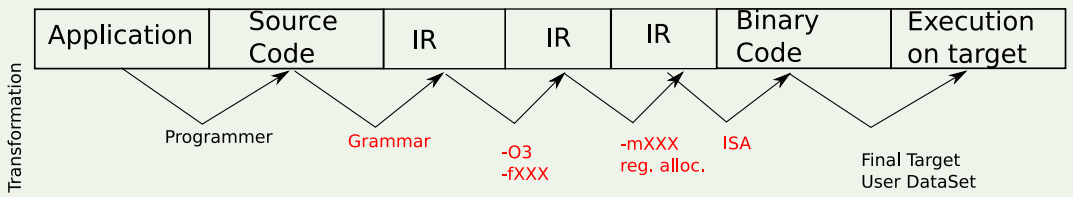
- SSA form : program as transformable data. Program transformation : parallelization, vectorization ...
- Register allocation.
- Instruction scheduling : based on data type arithmetics
- Asumptions about target
- Pattern matching for low level instructions selection

Illustration



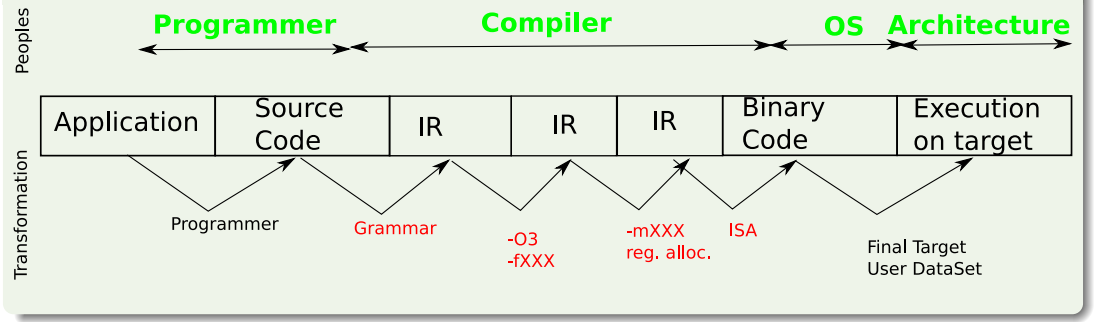
Compiler Support : Compiler And Architecture Links

Links between application and architecture



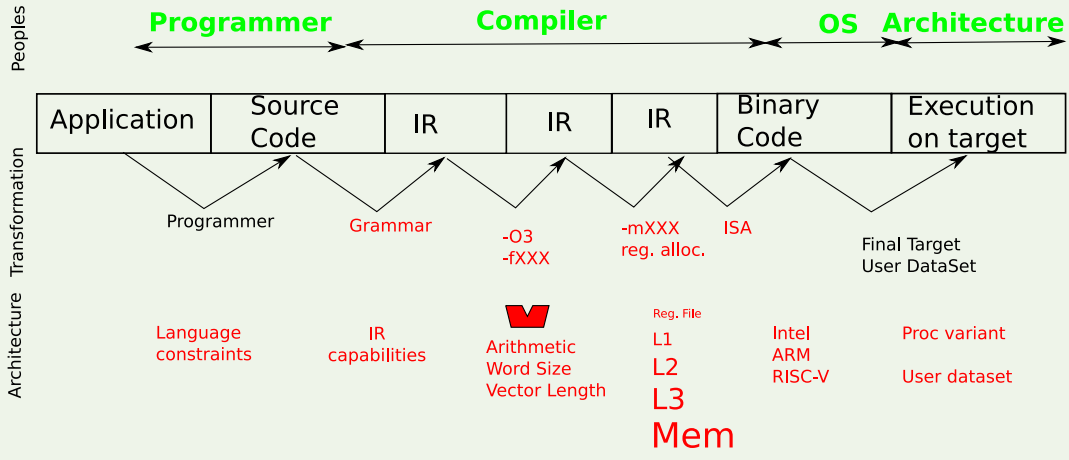
Compiler Support : Compiler And Architecture Links

Links between application and architecture



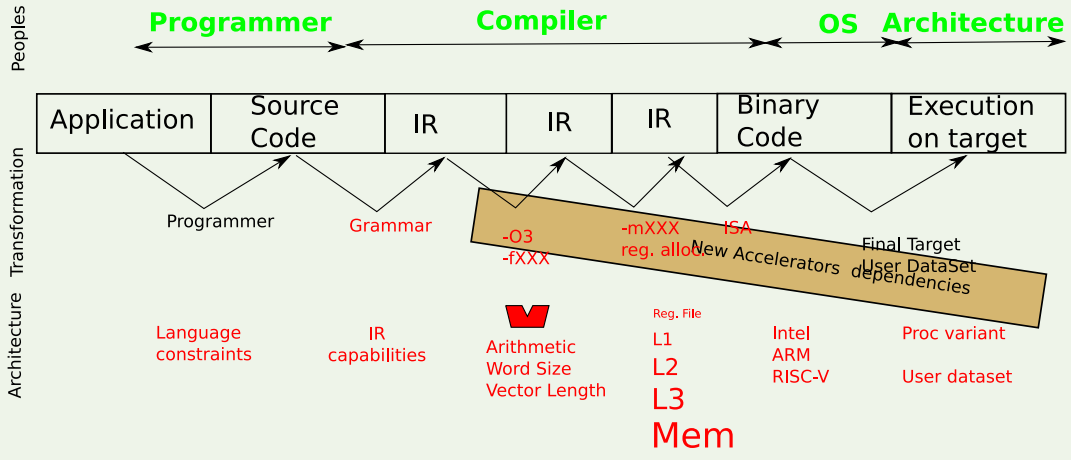
Compiler Support : Compiler And Architecture Links

Links between application and architecture



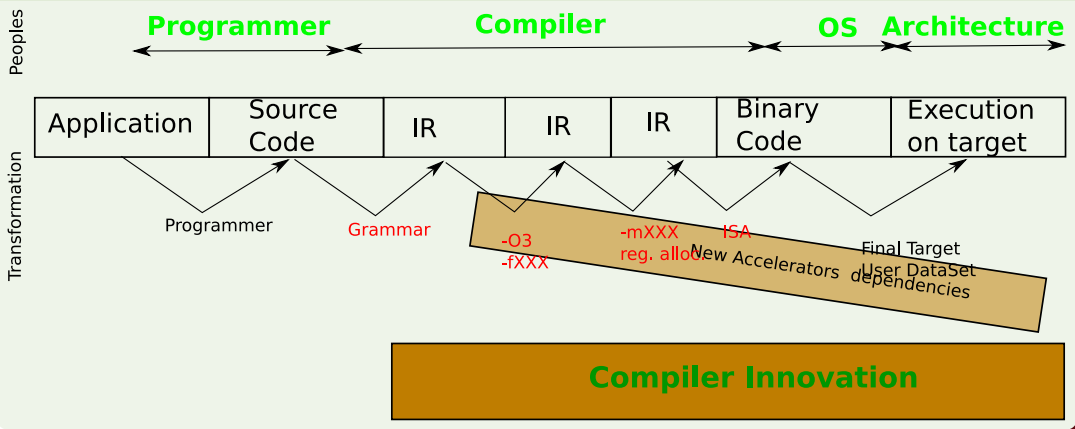
Compiler Support : Compiler And Architecture Links

Links between application and architecture



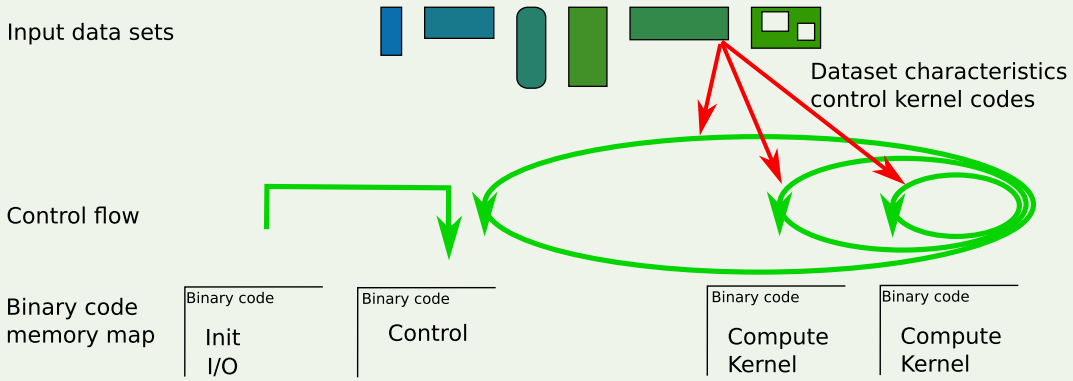
Compiler Support : Compiler And Architecture Links

Links between application and architecture



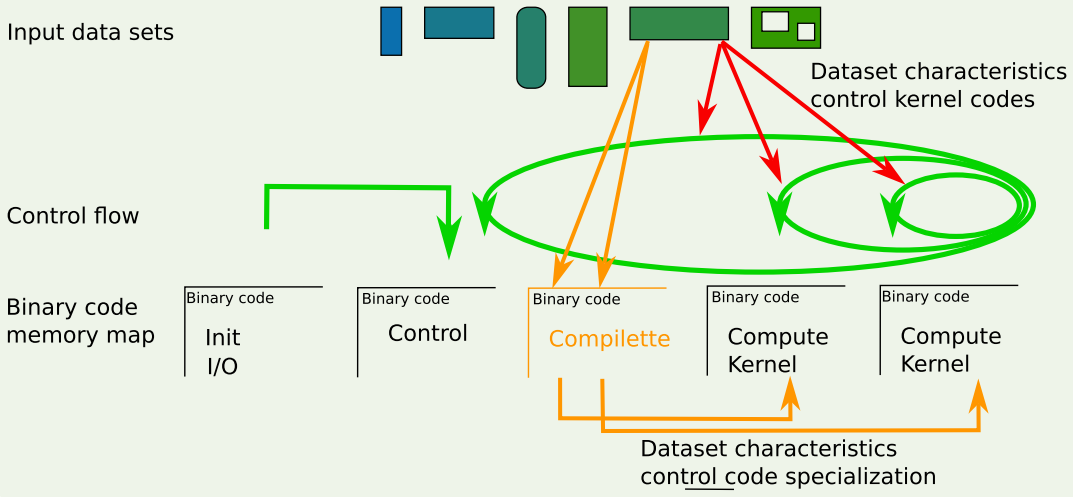
Compilette principle : “Working Example”

Control flow in application



Compilette principle : “Working Example”

Control flow in application with dynamic adaptation



List of Code Generation Scenarios

Compilation scenarios

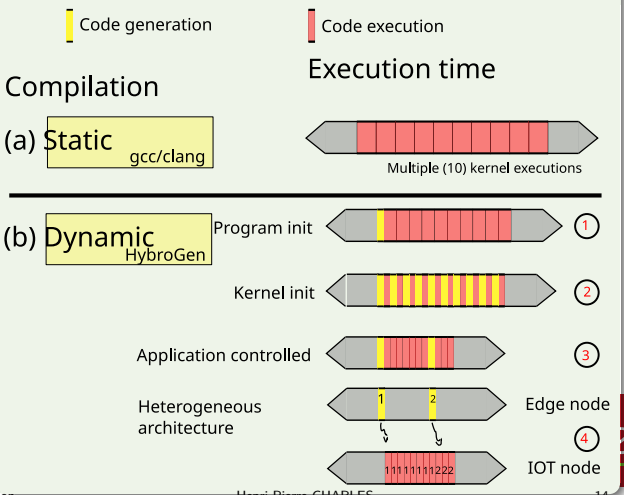
- (a) Static compilation
- (b) Dynamic adaptation
  1. Program initalization
  2. Kernel initialization
  3. Application controlled
  4. Heterogeneous architecture (multi-isa support)

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

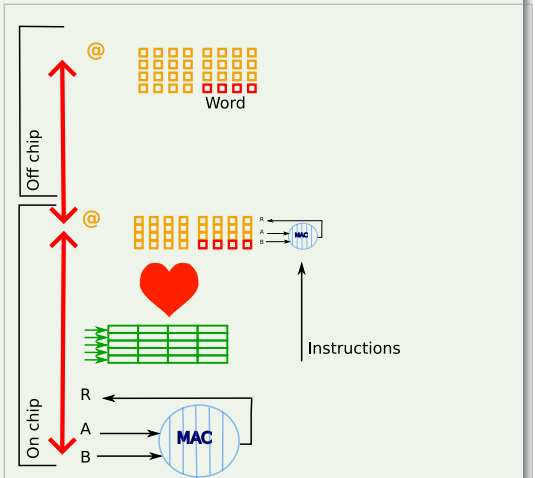
```
typedef  int (*pifi)(int);

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

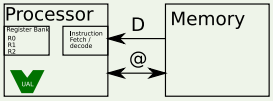
- Single program flow
- Non Von Neumann model : CPU send instructions to CSRAM
- DSL approach, which express
  - Heterogeneous computation (DONE)
  - Memory hierarchy (ONGOING)

Software support

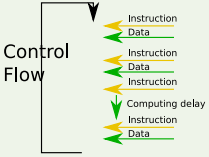
- HybroLang compiler `https://github.com/CEA-LIST/HybroGen`
- Functionnal emulator (based on QEMU): `https://github.com/CEA-LIST/csram-qemu-plugin`

Inverted Von Neumann Programming Model

Choosen Programming model



Bus transactions Code

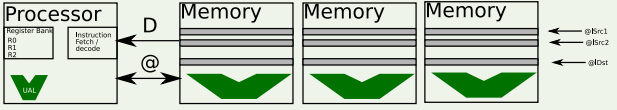


ld r0 = @A

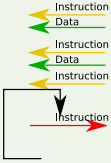
ld r1 = @B

add r2 = r0, r1

st r2 = @C



Bus transactions Code



ld r0 = @ISrc1

ld r1 = @ISrc2

ld r2 = @IDst

add IDst = ISrc1, ISrc2

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

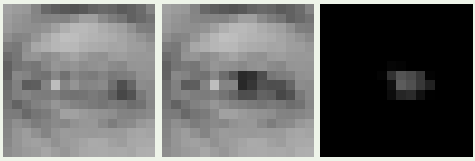
- Dynamic interleaving
- Instruction generator generator notion

HybroGen : ImageDiff-Run

CxRAM Usage

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

Dataset

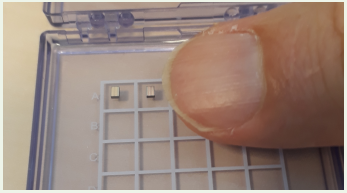


CxRAM-Status : Circuit

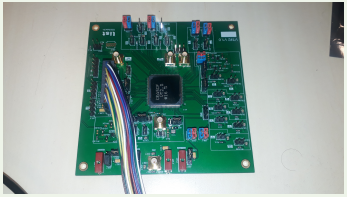
Chip design evolution

- 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