Compilers Challenges and dynamic applications

Interesting things are a low level

Henri-Pierre CHARLES

CEA DSCIN department / Grenoble

ven. 19 avril 2024

https://www.twitch.tv/hachepaisait

Introduction : Schedule

Schedule

- Speaker Presentation : 2mn
- Affiliation presentation : 2mn
- Usefull compiler notions : 5mn
- Compiler topics : 25 mn

Abstract

Compilers are rock solid piece of software (mostly) since the 80'. During the "Dennard scaling" era the compilers domain challenges where to integrate new applications needs and integrate micro-architecture hardware evolution (pipelines, vector ALU, cache hierarchies, special instructions, etc).
The effect of the end of the "Dennard scaling" (around 2006) has implied that performance evolution should be found in other directions. One major trend is based on hardware heterogeneity (big.LITTLE, CPU+DSP, CPU+FPGA, CPU+GPUs, etc).
In other hand applications become more and more dynamic and put pressure on memory hierarchy (indirect access, sparse computation). Dense computation can nearly reach the peak performance on modern processor (see TOP500 / Linpack), sparse computation can only use few percent of the peak performance.
I'll show some possible direction in compilation research to take advantage of the application dynamicity for code optimization.

Introduction : WhoAmI

Academic CV

- 1993 : PhD on compilers : instruction and data cache optimizations in GCC 2.95 / intel i860 (Lyon, France)
- 2008 : HdR on compilers : optimization on multimedia applications, dynamic application / itanium era (Versailles, France)
- 2012 : Research director CEA (Grenoble, France)

How to reach me

Main research topic : .. compilers !

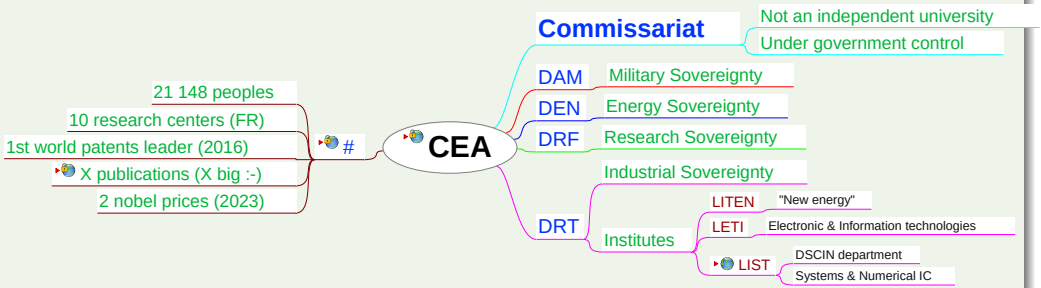
Make application auto-adaptive depending on datasets and computing architecture

Prof. Positions

- 1993 : Assistant professor Université de Versailles Saint-Quentin en Yvelines
- 2010 : Research engineer, CEA Grenoble
- 2012 : Research Director, CEA Grenoble
- 2023 : Fellow, CEA Grenoble

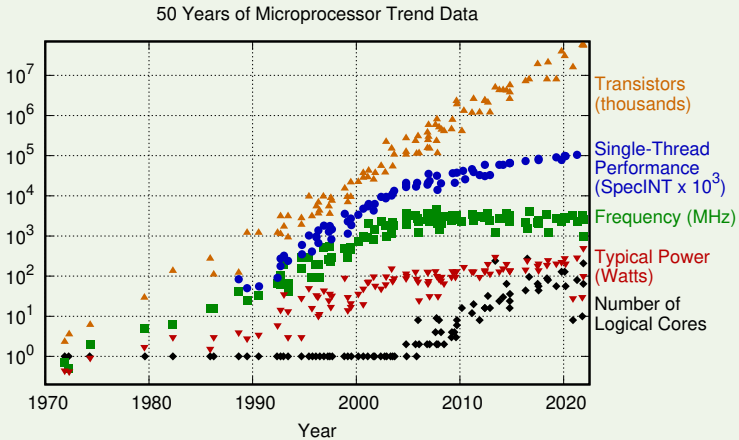
Introduction : CEA research centers

Alternative Energies and Atomic Energy Commission



Introduction : TrendData

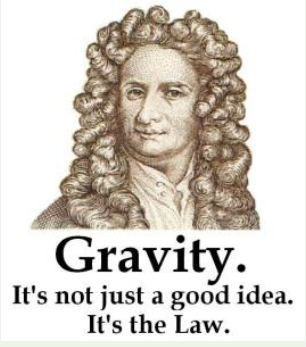
https://github.com/karlrupp/microprocessor-trend-data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Laws Vocabulary

Argumentation



Short list of importants laws / notions

- 1967 : Amdahl's law
- 1965 : Moore's law
- 1974 : Dennard scaling (Frequency evolution)
- 1995 : Wirth's law
- 2008 : Roofline model
- Memory bound IO bound
- CPU bound CPU bound

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

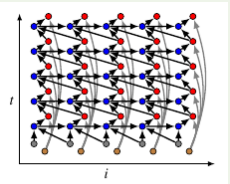
- “The Pascal User Manual and Report”
- Algorithms + Data Structures = Programs
- Wirth’s law (1995) “Software is getting slower more rapidly than hardware is becoming faster.” Article "A Plea for Lean Software"

Scientific Evolution : Compilation Research Domains

Compilation Research Topics Map

- Find code structure: Extract parallelism : polyhedral approach
- Assertion on legacy code: correctness proof, hard realtime, model checking
- Security: HW attach counter mesure, obfuscation
- Tools for scalability: Systems and library for big parallel machines
- Reproductibility: Statistics tools and reproductible research
- Ad hoc code optimization: Application driven code optimization
- Legacy compiler optimization: follow the HW evolution
- New code generation paradigm: JIT, Dynamic code generation

Polyhedral model



(a) CDAG for $T = 4, N = 7$

Algorithmic accelerator

Usefull Compiler Notions : What Is Compiler Support ?

Usual events in compiler designer life

- New hardware architecture :
  - 2001 : Intel Itanium
  - 2007 : Sony, Thoshiba, IBM Cell (processor)
  - 2014 : RISC-V
  - New accelerator for ...
- New application domain
  - 2003 : Advanced Video Coding
  - 2005 : IA TensofFlow
- New optimization / idea
  - JIT (1999) with Java HotSpot (virtual machine)
  - New IR : LLVM (2002) `https://llvm.org/pubs/`
  - Compilettes (2004): "Efficient data driven run-time code generation"

Compiler support levels for a new architecture / accelerator

0. No support : assembly level or asm intrinsic + C
1. Possible link from instruction to compiler intermediate representation
2. Compiler optimization thanks to math properties
3. Large optimization domain open to research (kernel loops, tiling, unrolling, etc)
4. New compilation paradigms

UsefullCompilerNotions : UsefullTools

Usefull tools

- `https://qemu.org` Swiss Knife for hetetogeneous platform
- gprof : analyse code
- Performance counters
- Always known your opponents: read databooks
  - RISC-V `https://riscv.org/technical/specifications/`
  - ARM V8 `https://documentation-service.arm.com/static/606ef2575e70d934bc69e1bf`
  - Intel
  - ../..

Used in my compiler

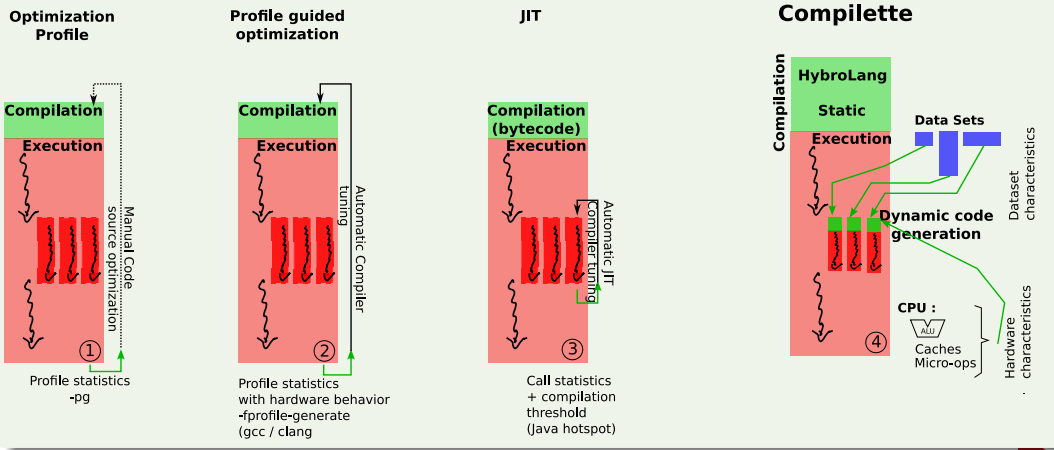
Compile time

- ANTLR : lexer / parser with style for Java / Python / C ...
- Postgresql : organize data
- Python : modern programming.

Run time None !

Compilers Topics : Code Generation Scenarios

Code Generation Scenarios : Manual, Profile Guided, JIT, Compilette Proposition



Compiler Topics : Initial Claim

Binary Code Generation should be Run-Time

- Static compilation fail to capture data-sets caracteritics
- Modern application are very dynamic
- Application run on multiple hardware

Why ? Because a modern software writer does not know

- What is the user data set
- What is the user processor brand / characteristics

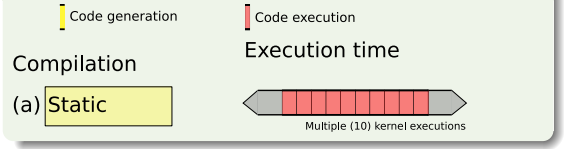
How ?

- Binary code should be generated at run-time depending on data characteristics
- Binary code generators should be small in size : Java JIT are MB in size
- Binary code generators should be fast : Java JIT is slow and is post-execution

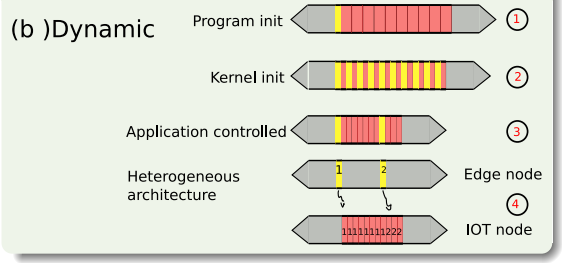
I want small & fast code generators, generated automatically ...
Generate generators ... Hmmm

Compiler Topics : Execution Scenarios

Static Code Generation



Dynamic Code Generation



HybroGen : Simple-Add : Generated Source

Code Macro instructions

```
#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_ADDI_RRI_I_32_1(r1,r0,i0) /* ADD */ \
do { \
        riscv_G32(((i0 & 0xfff) << 20)|((r0 & 0x1f) << 15)|((0x0 & 0x7) << 12)|((r1 & 0x1f) << 7
} while(0)
```

HybroGen : Simple-Add-Generated

Instruction Selector

```
void riscv_genADD_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 == REGISTER && P1.
    {
        RV32I_ADDI_RRI_I_32_1(P0.regNro, P1.regNro, P2.valueImm);
    }

    else if ((P0.arith == 'i') && (P0.wLen <= 32) && (P0.vLen == 1) && P0.ValOrReg == REGISTER &
    {
        RV32I_ADD_RRR_I_32_1(P0.regNro, P1.regNro, P2.regNro);
    }
```

HybroGen : Simple-Add-Source

Simple Addition with specialization

```
int main(int argc, char * argv[])
{
  h2_insn_t * ptr;
  int in0, in1, res;
  pifi fPtr;

  if (argc < 3)
    {
      printf("Give␣2␣values\n");
      exit(-1);
    }
  in0  = atoi (argv[1]);   // Get the users values in1 & in2
  in1  = atoi (argv[2]);
  ptr  = h2_malloc (1024);  // Allocate memory for 1024 instructions
  printf("//␣Compilette␣for␣simple␣addition␣between␣1␣variable␣with\n");
  printf("//␣code␣specialization␣on␣value␣=␣%d\n", in0);
  fPtr = (pifi) genAdd (ptr, in0); // Generate instructions
  res  = fPtr(in1);  // Call generated code
  printf("%d␣+␣%d␣=␣%d\n", in0, in1, res);
```

HybroGen : Simple-Add-Source

Simple Addition with specialization

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

h2_insn_t * genAdd(h2_insn_t * ptr, int  b)
{
  #[
  int 32 1 add (int 32 1 a)
  {
    int 32 1 r;
    r = #(b) + a; // b values will be included in code generation
    return r;
  }
  ]#
  return (h2_insn_t *) ptr;
}
```

HybroGen : Simple how to Build

How to run example

```
gre061041:CodeExamples/>./RunDemo.py -a riscv -i Add-With-Specialization
Namespace(arch=['riscv'], clean=False, debug=False, inputfile=['Add-With-Specialization'], verbo
-->rm -f Add-With-Specialization Add-With-Specialization.c
-->which riscv32-unknown-elf-gcc
-->../HybroLang.py --toC --arch riscv --inputfile Add-With-Specialization.hl
-->riscv32-unknown-elf-gcc -Wall -o Add-With-Specialization Add-With-Specialization.c
('3', '25')
-->qemu-riscv32 Add-With-Specialization 3 25
gre061041:CodeExamples/>qemu-riscv32 Add-With-Specialization 3 25
// Compilette for simple addition between 1 variable with
// code specialization on value = 3
3 + 25 = 28
```

HybroGen : Simple-Add Generated code

Main code generator

```
h2_insn_t * genAdd(h2_insn_t * ptr, int  b)
{
/* Code Generation of 4 instructions */
/* Symbol table :*/
        /*VarName = { ValOrLen, arith, vectorLen, wordLen, regNo, Value} */
        h2_sValue_t a = {REGISTER, 'i', 1, 32, 10, 0};
        h2_sValue_t h2_outputVarName = {REGISTER, 'i', 1, 32, 10, 0};
        h2_sValue_t r = {REGISTER, 'i', 1, 32, 5, 0};
        h2_sValue_t h2_00000003 = {REGISTER, 'i', 1, 32, 6, 0};


/* Label  table :*/
#define riscv_genLABEL(LABEL_ID) labelAddresses[LABEL_ID] = h2_asm_pc;
h2_insn_t   * labelAddresses []={
        };


        h2_asm_pc = (h2_insn_t *) ptr;
```

HybroGen : Simple-Add Generated Code

Compilette

```
h2_asm_pc = (h2_insn_t *) ptr;
h2_codeGenerationOK = true;
riscv_genMV_2(h2_00000003, (h2_sValue_t) {VALUE, 'i', 1, 32, 0, (b)});
riscv_genADD_3(r, h2_00000003, a);
riscv_genMV_2(h2_outputVarName, r);
riscv_genRET_0();
/* Call back code for loops */
h2_save_asm_pc = h2_asm_pc;
h2_asm_pc = h2_save_asm_pc;
h2_iflush(ptr, h2_asm_pc);
```

HybroGen : Simple-Add-Exec

Simple run

```
gre061041:CodeExamples/>qemu-riscv32 Add-With-Specialization 3 40
// Compilette for simple addition between 1 variable with
// code specialization on value = 3
3 + 40 = 43
```

Run with debug

```
qemu-riscv32 Add-With-Specialization 3 25
// Compilette for simple addition between 1 variable with
// code specialization on value = 3
0x19008 : RV32I_MV_RI_I_32_1
0x1900c : RV32I_ADD_RRR_I_32_1
0x19010 : RV32I_MV_RR_I_32_1
0x19014 : RV32I_RET__I_32_1
3 + 25 = 28
```

HybroGen : Simple-Add Debug

1 shell to Run / interact

```
gre061041:CodeExamples/>qemu-riscv32 -g \
    7777 Add-With-Specialization 3 25
// Compilette for simple addition
// between 1 variable with
// code specialization on value = 3
0x19008 : RV32I_MV_RI_I_32_1
0x1900c : RV32I_ADD_RRR_I_32_1
0x19010 : RV32I_MV_RR_I_32_1
0x19014 : RV32I_RET__I_32_1
```

1 shell to Debug / observe

```
riscv32-unknown-elf-gdb Add-With-Specialization
GNU gdb (GDB) 9.2
../..
(gdb) target remote :7777
(gdb) break main
Breakpoint 1 at 0x107c6: file Add-With-Specialization.c, line 201.
(gdb) c
Continuing.
Breakpoint 1, main (argc=3, argv=0x40800374) at Add-With-Specializat
201         if (argc < 3)
(gdb) n
206         in0  = atoi (argv[1]);
// Get the users values in1 & in2
211         fPtr = (pifi) genAdd (ptr, in0); // Generate instructions
(gdb)
212         res  = fPtr(in1);  // Call generated code
(gdb) x/4i fPtr
   0x19008:     ori     t1,zero,3
   0x1900c:     add     t0,t1,a0
   0x19010:     mv      a0,t0
   0x19014:     ret
(gdb)
```

HybroGen : More Complex Example

Original Code

```
void VectorMatrixProduct (vector_t out, vector_t in, matrix_t mat)
{
  for (int column = 0; column < T; column++)
    {
      out[column] = 0;
      for (int line = 0; line < T; line++)
          out[column] += mat[line][column] * in[line];
    }
}
```

Data

$$\begin{bmatrix} S_x & 0 & 0 & T_x \\ 0 & S_y & 0 & T_y \\ 0 & 0 & S_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Optimized code based on dataset

```
void VectorMatrixProductUnrolledSR(vector_t o, vector_t i, matrix_t m)
{
  o[0] = m[0][0]*i[0] +                          + m[3][0]*i[3];
  o[1] =              m[1][1]*i[1] +             + m[3][1]*i[3];
  o[2] =                           m[2][2]*i[2] + m[3][2]*i[3];
  o[3] =                                        +        i[3];
}
```

Conclusion : Working On

Already done

- Compilation from computing in memory architecture (C-SRAM)
- Compilation for IA application on C-SRAM
- Transprecision support for classical architectures

On going

- Value based optimization
- Transprecision support for variable precision hardware
- Complex applications using transprecision

Future works

- Continue to work on computing in memory
- Optimize for applications
  - Graphic / image processing
  - Network applications
  - Code specialization in gmp / mpfr
  - So many applications :-)
- Optimize for new accelerators
- Re-start workshop on dynamic compilation ?
- Start a “Compilation Channel” on twitch to become rich, become famous, live in Dubai ! Maybe not :-)

Conclusion : Last words (for today)

Compilation is forever

- Programming Languages are stable for long period
- DSL Domain Specific Languages

Compilation technologies are usefull

- Transform data-sets
- Search in datas
- Rewrite / generate texts

New paradigms / new objectives

- Manage Hardware heterogeneity. Intel is no more the only processor brand : ARM, Risc-V, Embedded platforms
- Compilers should help to “remove layers” i.e. generate simpler code, not add a new layer

Links

- https://hpcharles.wordpress.com/
- https://github.com/CEA-LIST/HybroGen