# 毕昇编译器设计与优化技术

魏伟　华为编译器与编程语言实验室

2021-04-17

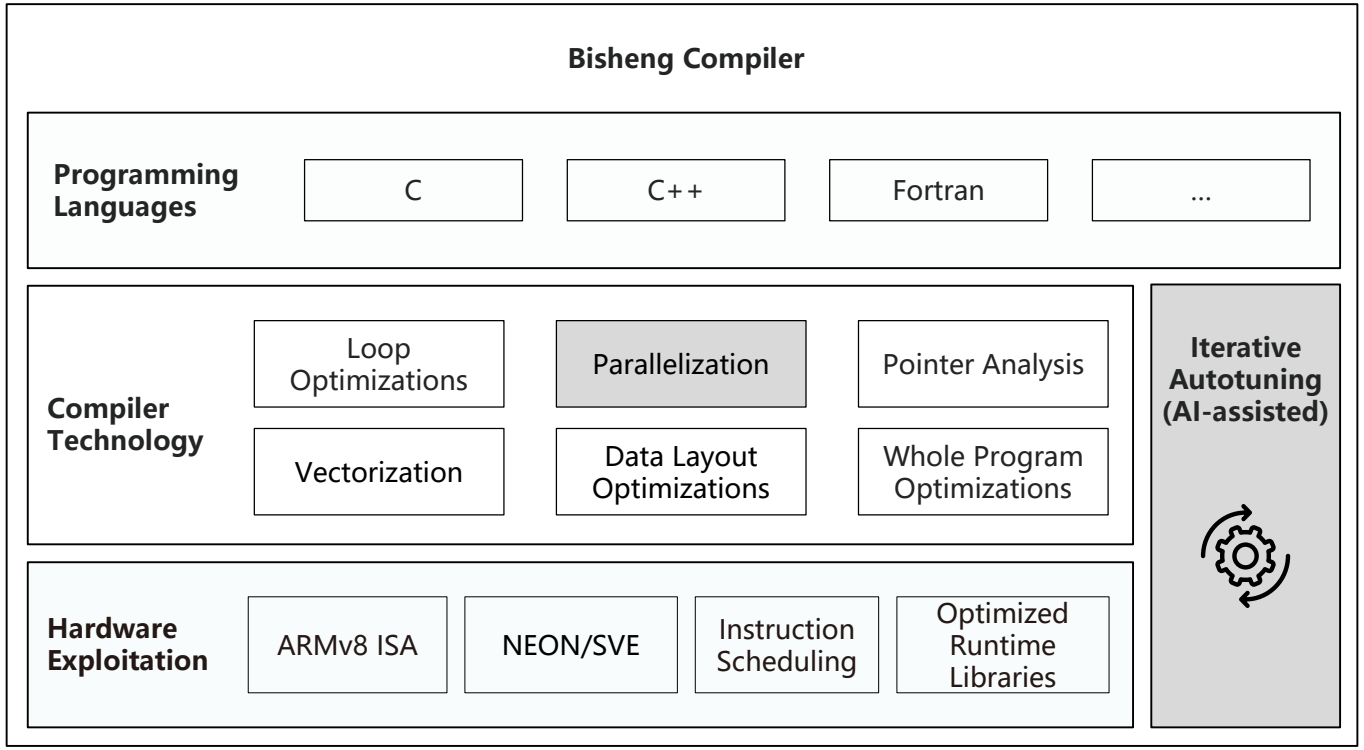
HUAWEI

# 目录



HUAWEI

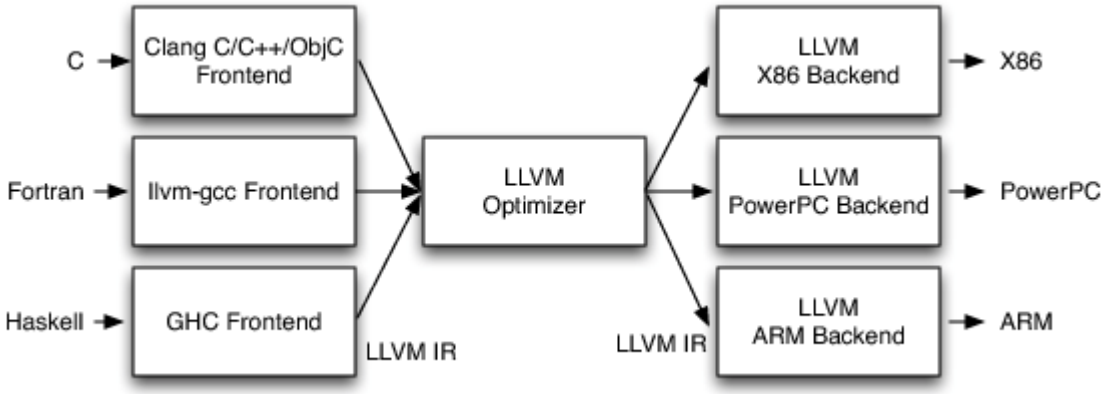
# 毕昇编译器介绍




- High-performance compiler for Linux targeting Kunpeng 920 processor
- Based on open-source LLVM 10 and PGI Fortran frontend ("Flang") with in-house enhancements
- **Compilation Technology:** Enhanced vectorization, loop optimizations, instruction selection, etc.
- **Hardware Exploitation:** Leverage NEON/SVE instructions to optimize the throughput of generated code as well as runtime libraries.
- **ML-based Autotuning:** Search for the optimal combination of compilation parameters on fine-grained code regions and achieve the best possible performance.

# 毕昇编译器介绍：LLVM架构及IR

- LLVM's Implementation of Three-Phase Design.
- A front end is responsible for parsing, validating and diagnosing errors in the input code, then translating the parsed code into LLVM IR.
- LLVM IR is optionally fed through a series of analysis and optimization passes which improve the code, then is sent into a code generator to produce native machine code.



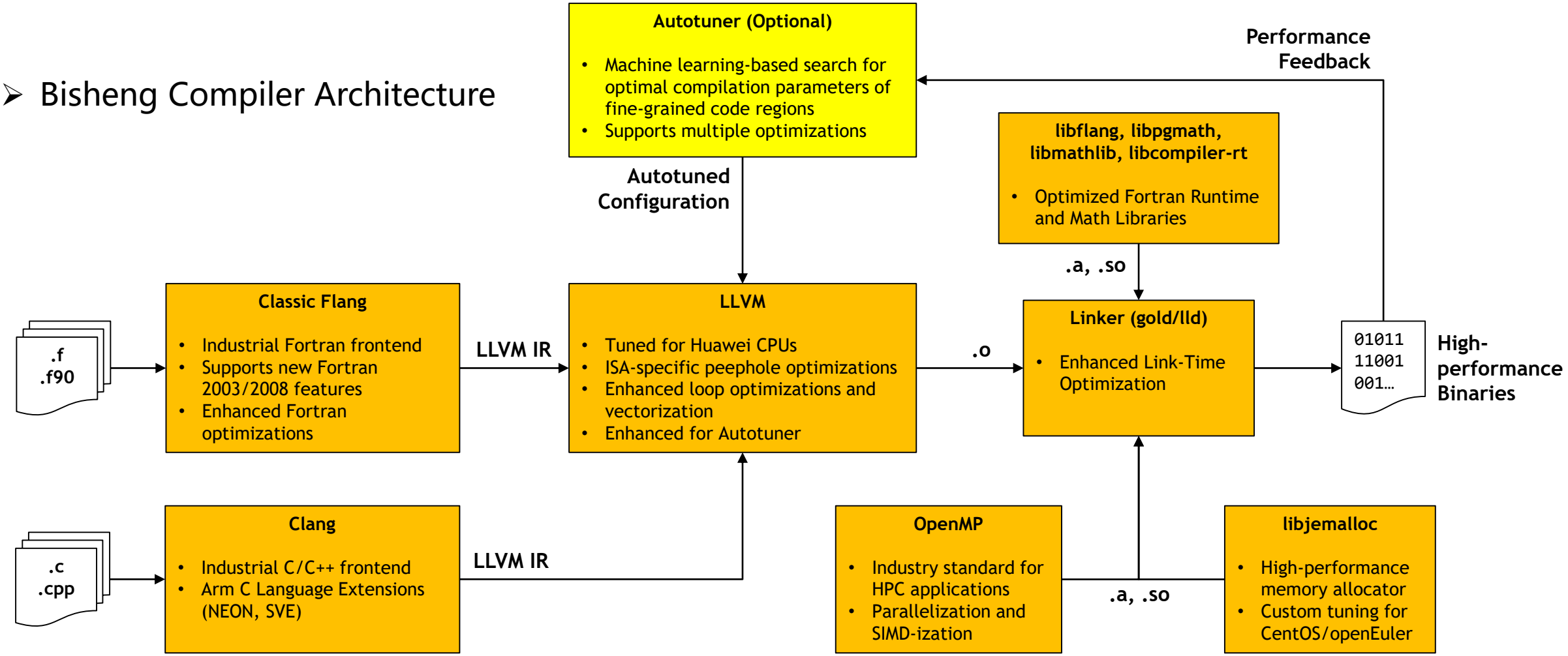

- LLVM's Code Representation: LLVM IR.
- A low-level RISC-like virtual instruction set.
- Three address form, strongly typed.

```
unsigned add1(unsigned a, unsigned b) {
  return a+b;
}

// Perhaps not the most efficient way to add two numbers.
unsigned add2(unsigned a, unsigned b) {
  if (a == 0) return b;
  return add2(a-1, b+1);
}
```

```
define i32 @add1(i32 %a, i32 %b) {
entry:
  %tmp1 = add i32 %a, %b
  ret i32 %tmp1
}

define i32 @add2(i32 %a, i32 %b) {
entry:
  %tmp1 = icmp eq i32 %a, 0
  br i1 %tmp1, label %done, label %recurse

recurse:
  %tmp2 = sub i32 %a, 1
  %tmp3 = add i32 %b, 1
  %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
  ret i32 %tmp4

done:
  ret i32 %b
}
```

# 毕昇编译器介绍：基本架构

- Bisheng Compiler Architecture



# 目录



HUAWEI

# 毕昇编译器优化技术：An interesting example

```
static const unsigned long long magic = 0x03f08c5392f756cdULL;

static const int table[64] = {
     0,  1, 12,  2, 13, 22, 17,  3,
    14, 33, 23, 36, 18, 58, 28,  4,
    62, 15, 34, 26, 24, 48, 50, 37,
    19, 55, 59, 52, 29, 44, 39,  5,
    63, 11, 21, 16, 32, 35, 57, 27,
    61, 25, 47, 49, 54, 51, 43, 38,
    10, 20, 31, 56, 60, 46, 53, 42,
     9, 30, 45, 41,  8, 40,  7,  6,
};

int
myctz (unsigned long long b) {
    unsigned long long lsb = b & -b;
    return table[(lsb * magic) >> 58];
}
```

```
myctz:                                  // @myctz
// %bb.0:                               // %entry
        mov     x9, #22221
        movk    x9, #37623, lsl #16
        neg     x8, x0
        and     x8, x8, x0
        movk    x9, #35923, lsl #32
        movk    x9, #1008, lsl #48
        mul     x8, x8, x9
        adrp    x9, table
        add     x9, x9, :lo12:table
        lsr     x8, x8, #56
        and     x8, x8, #0xfc
        ldr     w0, [x9, x8]
        ret
```

```
myctz:
.LFB0:
        .cfi_startproc
        rbit    x0, x0
        clz     x0, x0
        and     w0, w0, 63
        ret
        .cfi_endproc
```

**Deepsjeng --** an Artificial Intelligence: alpha-beta tree search (Chess).

Algorithm　Count number of trailing zeros using a de Bruijn cycle

《Hacker’ s Delight》(Second Edition)

**RBIT**

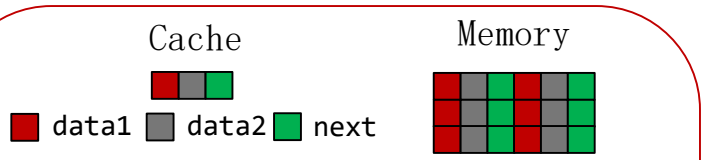
Reverse Bits reverses the bit order in a register.

**CLZ**

Count Leading Zeros returns the number of binary zero bits before the first binary one bit in a value.

# 毕昇编译器优化技术： Structure reorganization

- Whole-program optimizations to improve cache utilization
- Transform an array of structures into a structure of arrays
- Structures can either be explicit or they can be inferred by inspecting array usage inside loops

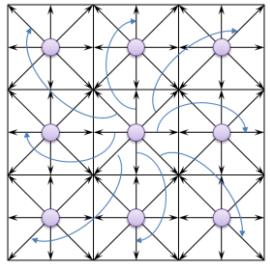


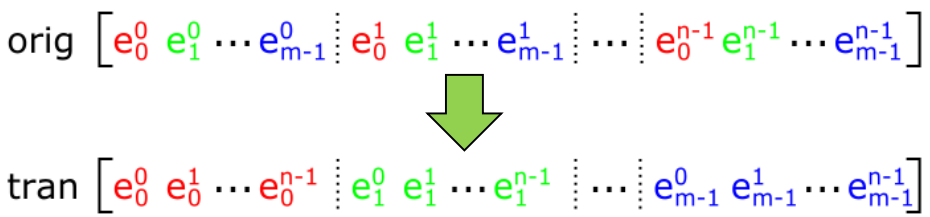

```
typedef struct cand_struct {
  int data1;
  int data2;
  struct cand_struct *next;
} cand_struct_t;

cand_struct_t *array =
  (cand_struct_t*)calloc(N, …);

cand_struct_t *tmp = array;
while (tmp) {
  sum += tmp->data1;
  tmp = tmp->next;
}
```


```
void *array = calloc(N, …);
int *data1_arr = (int*)array;
int *data2_arr = data1_arr + N;
unsigned *next_idx_arr = data2_arr + N;
memset(next_idx_arr, -1, N*sizeof(unsigned));

unsigned idx = 0;
while (idx != -1) {
  sum += data1_arr[idx];
  idx = next_idx_arr[idx];
}
```

# 毕昇编译器优化技术： Array Restructuring

- Programs access arrays in fixed patterns which have poor cache locality
- In the LBM hotspot, each iteration reads 20 consecutive elements from srcGrid, and computes 20 non-consecutive elements in dstGrid
- Analyze array access pattern to find an index re-mapping formula that allows compiler to re-arrange the array to look like a structure of arrays
  - e.g. `orig[i]` ➔ `tran[ (i % m) * n + (i / m) ]`
- Rewrite all accesses to use the new indexing
- Profit from reduced cache misses vs. overhead from more complicated index computation





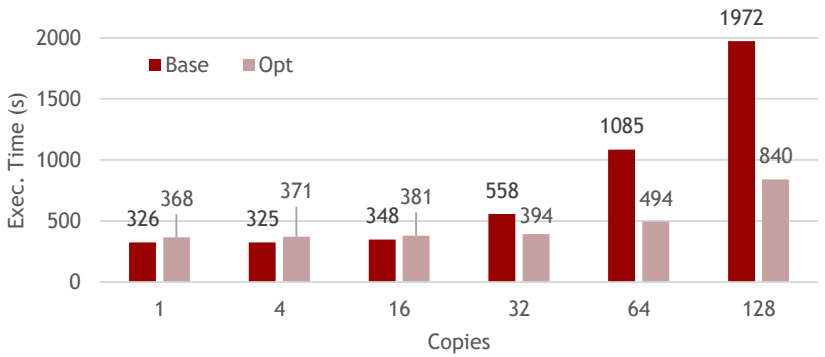

Lattice Boltzmann Method, LBM

**Before**
Stride 1 for srcGrid reads: good locality
Stride 20 for dstGrid writes: bad locality

| Index i | i = 0 | i = 20 | i = 40 |
|---|---|---|---|
| srcGrid | 0, 1, 2,... | 20, 21,22,... | 40, 41, 42,... |
| dstGrid | 0, -198, 202,... | 20, -178, 222,... | 40, -158, 242,... |

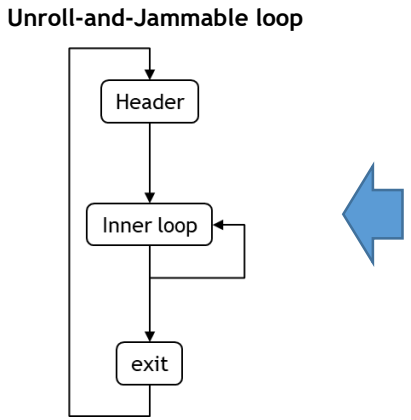
**After**
Stride 1 for all arrays: good locality
Hardware prefetcher can handle 20 streams

| Index i | i = 0 | i = 1 | i = 2 |
|---|---|---|---|
| srcGrid.g0 | 0 | 1 | 2 |
| srcGrid.g1 | 0 | 1 | 2 |
| ... | | | |
| srcGrid.g19 | 0 | 1 | 2 |
| dstGrid.g0 | 0 | 1 | 2 |
| dstGrid.g1 | -10 | -9 | -8 |
| dstGrid.g2 | 10 | 11 | 12 |
| ... | | | |



# 毕昇编译器优化技术： Loop Optimization

- A broad category of optimizations. Improves the performance of the loops through different measures.
- Improve cache utilization
- Reduce register pressure
- Reduce dynamic instruction count
- Reuse loaded/computed values.
- Expose opportunities for other optimizations e.g.
  - > Vectorization
  - > Instruction scheduling




## Loop Unroll and jam

Unrolling the outer loop. Then jamming (fusing) bodies of the inner loops.

- Helps better cache utilization

Residue loop not displayed in the example below

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < m; j++) {
    a[j] += b[i][j] + b[i-1][j];
  }
}
```

```
for (int i = 0; i < n-n%2; i+= 2) {
  for (int j = 0; i < m; j++) {
    a[j] += b[i][j] + b[i-1][j];
    a[j] += b[i+1][j] + b[i][j];
  }
}
```

## Loop Fusion/Distribution

These two optimization are opposite of each other. Loop fusion will merge bodies of two loops to create one, if possible. Distribution divides body of one loop to two separate loops.

How fusion can help:

- Reuse values
- Expose instruction scheduling opportunities.

How distribution can help:

- Reducing register pressure inside the loop.
- Expose vectorization opportunities.
- Expose opportunities for other loop optimizations.

In the example below, we would like to keep the first two statements together to reuse loaded values, but distribute the third statement which is not vectorizable. (Note that compiler might be able to partially vectorize the first loop, before distribution)

```
for (int i = m; i < n; i++) {
  a[i] = b[i-1] + b[i];
  c[i] = b[i-1] - b[i];
  d[i-1] = d[i-2] + 1;
}
```

```
for (int i = m; i < n; i++) {
  a[i] = b[i-1] + b[i];
  c[i] = b[i-1] - b[i];
}

for (int i = m; i < n; i++) {
  d[i-1] = d[i-2] + 1;
}
```

# 毕昇编译器优化技术： Loop Optimization

## Loop Unrolling

Replicating body of the loop, so fewer iterations are needed.

- Reduces dynamic instruction count.
- Exposes opportunities for values reuse and better scheduling.

```
for (int i = 0; i < n; i++) {
  a[i] = b[i-1] + b[i];
}
```

```
int ub = n - n%2;
for (int i = 0; i < ub; i += 2) {
  a[i] = b[i-1] + b[i];
  a[i+1] = b[i] + b[i+1];
}
if (i < n)
  a[i] = b[i-1] + b[i];
```

## Loop Unswitching

Hoisting invariant conditions out of loop and versioning the loop.

- Helps to expose other opportunities
- Reduces the number of branches executed.

```
for (int i = 0; i < n; i++) {
  a[i] = 1;
  if (b[j] > 10)
    a[i] += 1;
}
```

```
if (b[j] > 10)
  for (int i = 0; i < n; i++)
    a[i] = 1;
else
  for (int i = 0; i < n; i++)
    a[i] = 2;
```

```
void f (std::map<int, int> m)
{
  for (auto it = m.begin (); it != m.end (); ++it) {
    /* if (b) is semi-invariant. */
    if (b) {
      b = do_something();     /* Has effect on b */
    } else {
      /* No effect on b */
    }
    statements;               /* Also no effect on b */
  }
}
```

Loop split

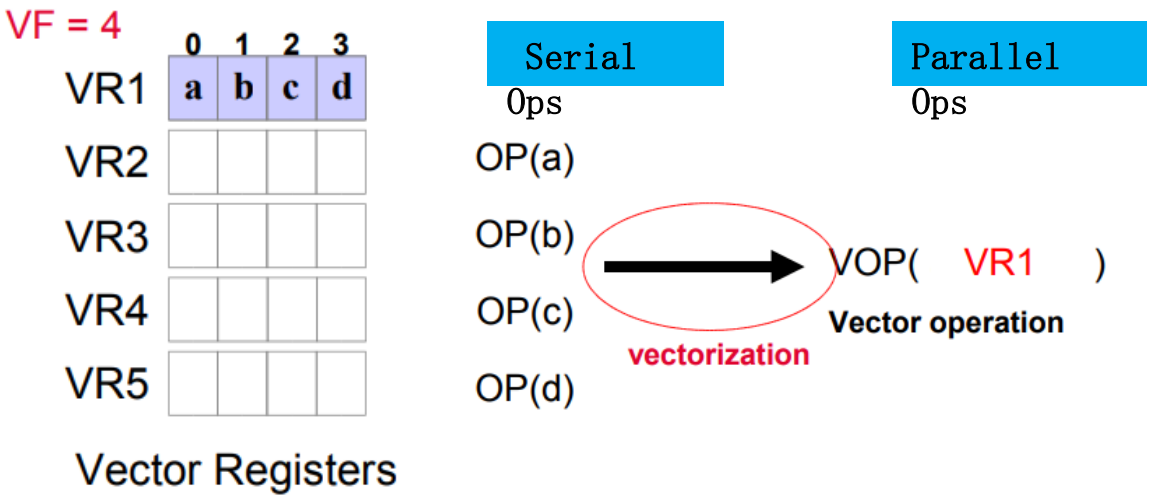
```
void f (std::map<int, int> m)
{
  for (auto it = m.begin (); it != m.end (); ++it) {
    if (b) {
      b = do_something();
    } else {
      ++it;
      statements;
      break;
    }
    statements;
  }
  for (; it != m.end (); ++it) {
    statements;
  }
}
```

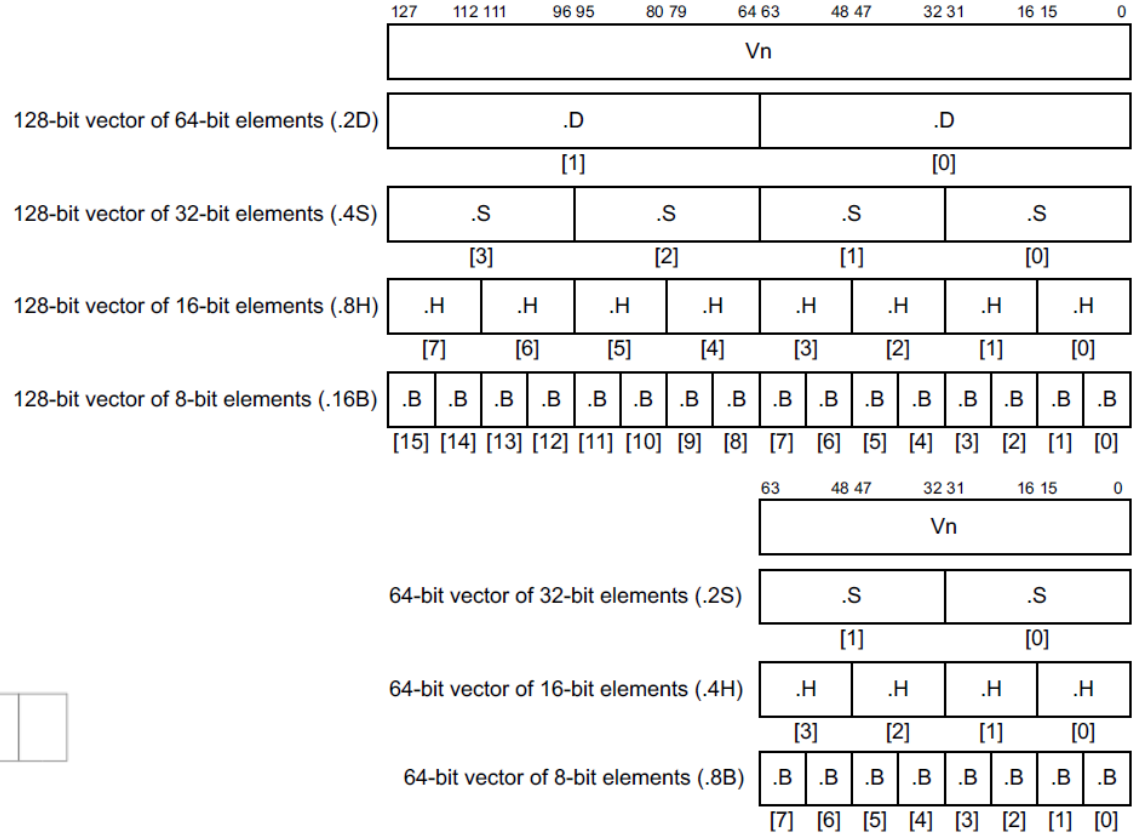
```
void f (std::map<int, int> m)
{
  for (auto it = m.begin (); it != m.end (); ++it);
}
```

Loop delete

```
void f (std::map<int, int> m)
{
}
```

# 毕昇编译器优化技术： Auto Vectorization




SIMD（Neon） vectors in AArch64

```
int a[N], b[N], c[N];

for (1..N, i++) {
  a[i] = b[i] + c[i];
}
```

```
int a[N], b[N], c[N];

for (1..N/4, i+=4) {
  a[i+0] = b[i+0] + c[i+0];
  a[i+1] = b[i+1] + c[i+1];
  a[i+2] = b[i+2] + c[i+2];
  a[i+3] = b[i+3] + c[i+3];
}
```

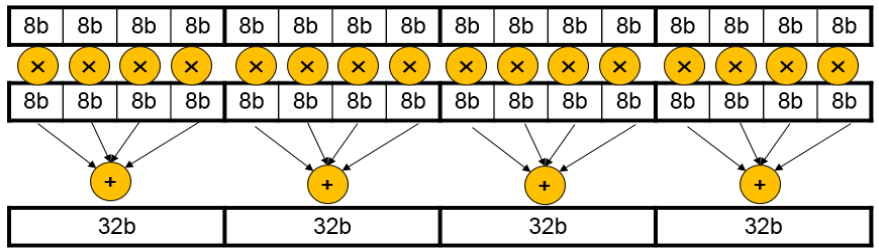
```
int a[N], b[N], c[N];

for (1..N/4, i+=4) {
  a[0..3] = b[0..3] + c[0..3];
}
```

# 毕昇编译器优化技术：Exploitation of NEON/SVE instructions

## Sum of absolute differences

```
int sum = 0;  // pix1 and pix2 are uint8_t
for (int x = 0; x < 16; x++)
  sum += abs(pix1[x] - pix2[x]);
```

```
movi    v1.16b,#1
ldr     q3, [x12, x0]
ldr     q2, [x13, x1]
uabd    v2.16b, v2.16b, v3.16b
udot    v0.4s, v2.16b, v1.16b
add     x8, x10, x0
add     x9, x11, x1
```



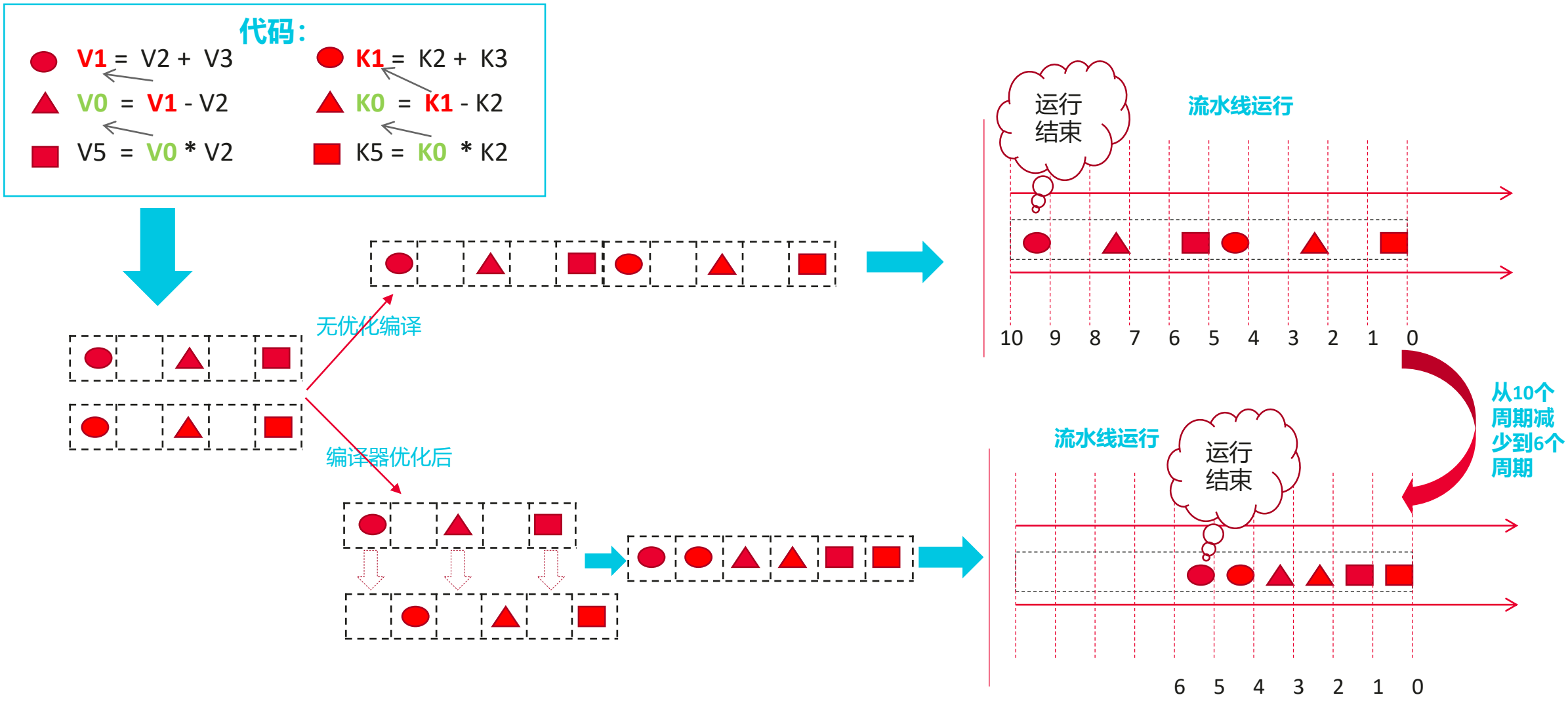

## Rounded halving add

```
for (int x = 0; x < i_width; x++)
  dst[x] = (src1[x] + src2[x] + 1) >> 1;
```

```
ldr     d0, [x2, x16]
ldr     d1, [x4, x16]
ushll   v0.8h, v0.8b, #0
ushll   v1.8h, v1.8b, #0
mvn     v0.16b, v0.16b
sub     v0.8h, v1.8h, v0.8h
shrn    v0.8b, v0.8h, #1
str     d0, [x0, x16]
```

```
ldr     d0, [x2, x16]
ldr     d1, [x4, x16]
urhadd  v0.8b, v1.8b, v0.8b
str     d0, [x0, x16]
```

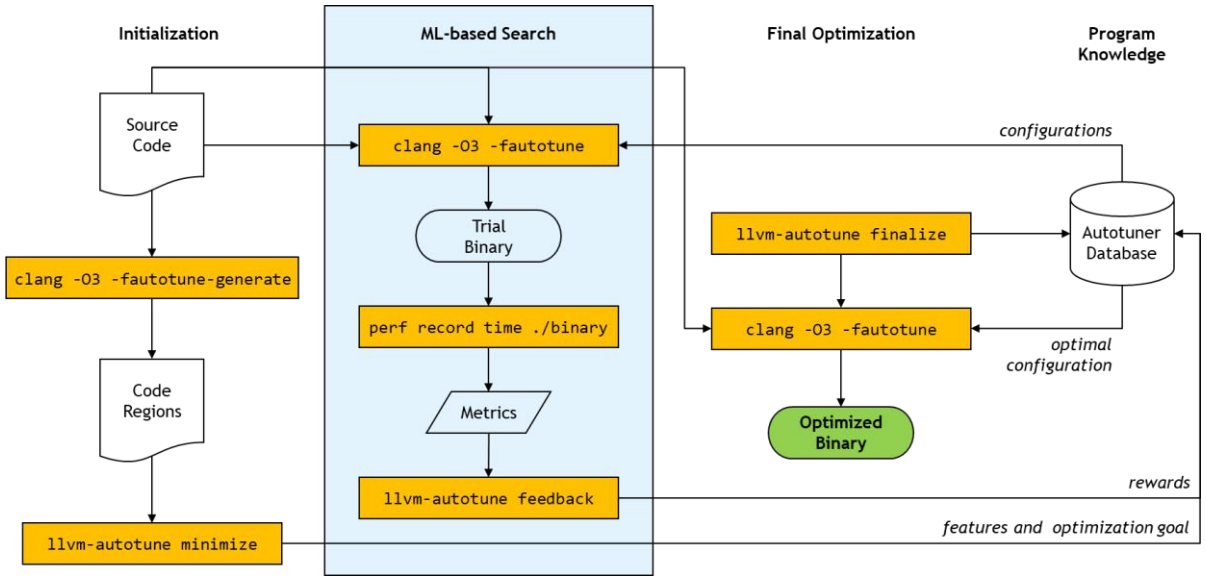
# 毕昇编译器优化技术：Pipeline优化，代码最佳执行效率




HUAWEI

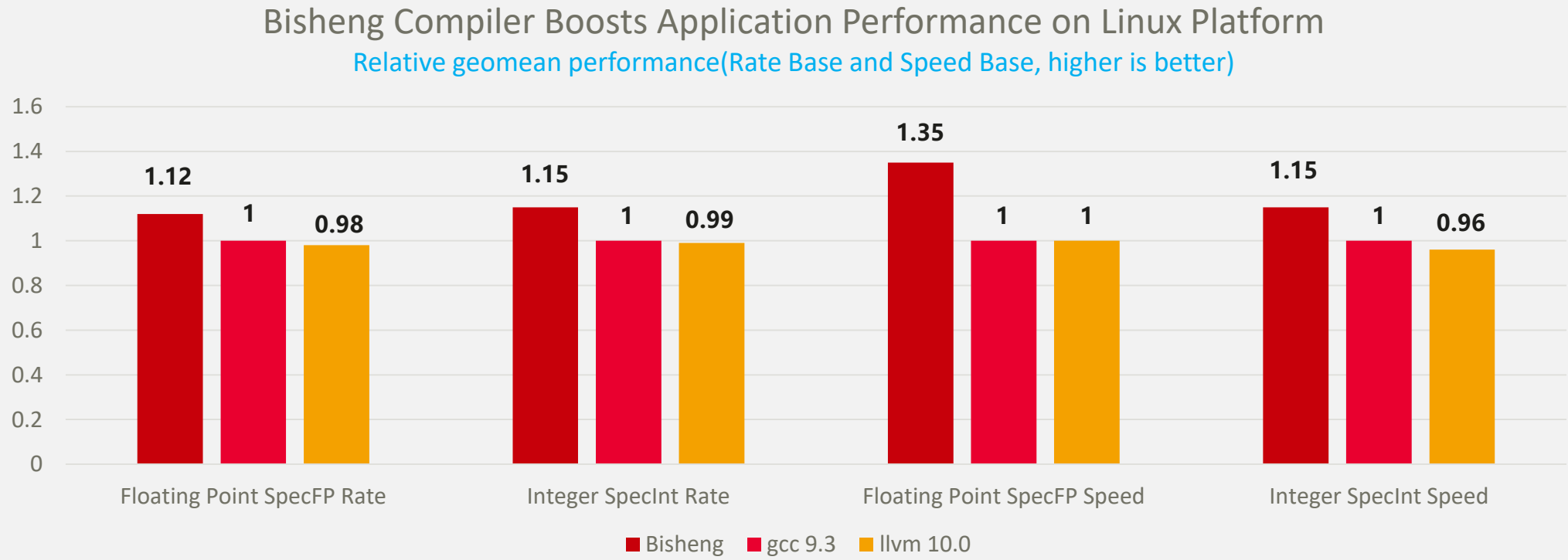
# 毕昇编译器优化技术： Autotuner

- Customize optimization strategy to fine-grained code regions (loops/functions); not restricted to global command-line options!
- Build search space by identifying code regions and profitable optimization passes, and use machine learning algorithms with feedback to find the best optimization parameters
  - e.g. loop unroll factor, inlining threshold, instruction scheduling parameters, optimizer pipeline
- Leverage profile feedback to speed up autotuning
- Store insight about software, compiler configurations, and performance effects in databases for better tuning results
- Employ data mining and machine learning to predict performance, prune iterative search space, optimize initial compilation, etc.
- Up to 8% speedup achieved in real-world applications



# 毕昇编译器：SPEC2017上性能较业界编译器平均高15%以上

毕昇编译器与鲲鹏芯片协同，通过编译器技术充分发挥芯片的性能，提升鲲鹏硬件平台上业务的性能体验。**基于鲲鹏上编译器优化，SPEC2017性能比gcc平均高15%以上。**




***SPEC作为业界芯片性能评分标准，SPEC的分数可以直观的体现出硬件的性能**

# 毕昇编译器：Fortran Frontend and Runtime Improvements

- Inline MINLOC/MAXLOC reduction intrinsics when profitable
  - +10% on 548.exchange2_r
- Inline NINT intrinsic
  - Hotspot in GRAPES; minor improvement
- More efficient memory allocation, heap-to-stack allocation
  - +3% on WRF
- Pow() call simplification
- Better interprocedural optimizations
  - Function specialization for PRIVATE procedures
  - VALUE promotion of INTENT(IN) arguments
- Directives: added on top of what Flang provided
  - OMP PARALLEL, OMP SIMD, UNROLL, VECTOR, PREFETCH, etc.
- Quad-precision floating-point math, higher array ranks

*Syntax*:

```
RESULT = MINLOC(ARRAY, DIM [, MASK] [,KIND] [,BACK])
RESULT = MINLOC(ARRAY [, MASK], [,KIND] [,BACK])
```

*Arguments*:

| | |
|---|---|
| *ARRAY* | Shall be an array of type INTEGER, REAL or CHARACTER. |
| *DIM* | (Optional) Shall be a scalar of type INTEGER, with a value between one and the rank of *ARRAY*, inclusive. It may not be an optional dummy argument. |
| *MASK* | Shall be an array of type LOGICAL, and conformable with *ARRAY*. |
| *KIND* | (Optional) An INTEGER initialization expression indicating the kind parameter of the result. |
| *BACK* | (Optional) A scalar of type LOGICAL. |

**F2008 Features**

| |
|---|
| **Coarrays** |
| **Data Declaration** |
| Maximum rank + corank = 15 |
| Quad-precision floating-point math |
| Allocatable components of recursive type |
| Implied-shape array |
| Data statement restrictions lifted |
| Type statement for intrinsic types |
| Declaring type-bound procedures |
| Value attribute is permitted for any nonallocatable nonpointer noncoarray |
| In a pure procedure the intent of an argument need not be specified if it has the value attribute |
| **Accessing data objects** |
| Omitting an ALLOCATABLE component in a structure constructor |
| Pointer function reference is a variable |
| **Intrinsic procedures and modules** |
| Bit manipulation |
| Optional argument RADIX added to SELECTED REAL KIND |
| Euclidean vector norms |
| Parity |
| **Programs and procedures** |
| Null pointer or unallocated allocatable as absent dummy arg. |
| Non pointer actual for pointer dummy argument |
| Generic resolution by procedureness |
| Generic resolution by pointer vs. allocatable |

# 毕昇编译器：资源获取及社区支持

| 交付件类型 | 链接 | 使用说明 |
| --- | --- | --- |
| 软件包 | https://www.huaweicloud.com/kunpeng/software/bishengcompiler.html | 鲲鹏社区下载后，解压使用 |
| sha256 | https://www.huaweicloud.com/kunpeng/software/bishengcompiler.html | 鲲鹏社区下载后，用于对比完整性校验结果 |
| 软件许可 | https://www.huaweicloud.com/kunpeng/software/bishengcompiler.html | 基于开放源码义务提供 |
| 文档 | https://www.huaweicloud.com/kunpeng/software/bishengcompiler.html | 毕昇编译器用户指南 |
| | https://support.huaweicloud.com/fg-autotuner-kunpengdevps/kunpengbisheng_20_0002.html | Autotuner特性指南 |
| 问题讨论 | https://bbs.huaweicloud.com/forum/thread-70301-1-1.html | 鲲鹏社区论坛，可发帖提问 |

# Q & A

HUAWEI

# Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

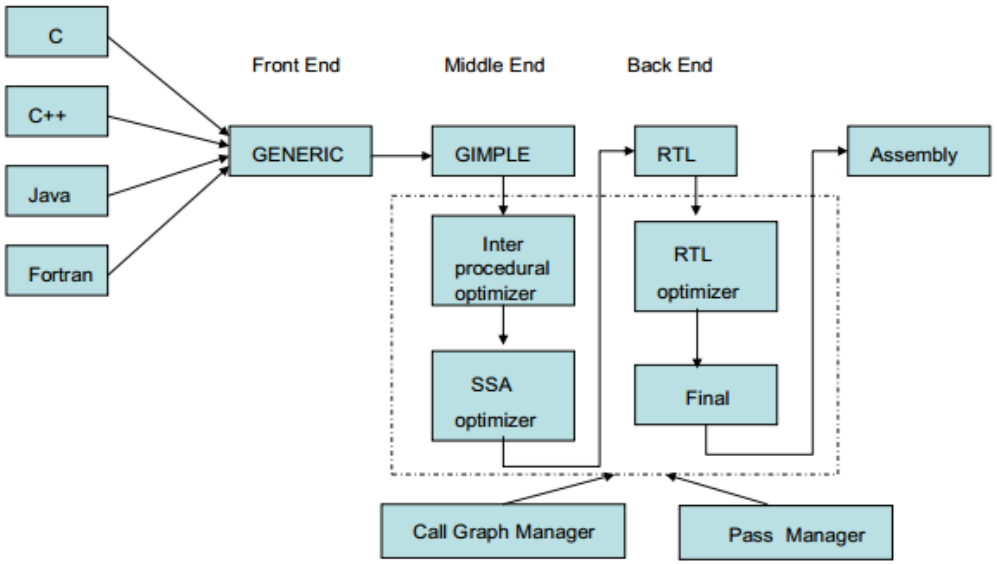


HUAWEI

# GCC编译器流程简介

- 编译过程

编译是将C/C++等语言转换为汇编语言的过程，其中主要包括词法分析、语法分析、机器无关优化、机器相关优化和汇编代码生成。编译器内部中间语言主要的几种形式是：GENERIC、GIMPLE、RTL，主要数据结构为TREE和RTX。




编译器是一种将高级语言转换为机器能识别的语言(机器码)的工具

- 输入：高级语言编写的源文件，如C、C++、JAVA、汇编
- 输出：机器能识别的二进制文件，如ELF文件
- 编译工具链组成：编译器、汇编器、链接器

# 鲲鹏GCC编译器介绍

鲲鹏GCC编译器基于开源GCC优化增强，充分发挥鲲鹏芯片计算性能，让应用在鲲鹏上跑得更快


HUAWEI

# 鲲鹏GCC资源获取及社区支持

**鲲鹏GCC二进制包官方下载地址：**

https://kunpeng.huawei.com/#/developer/devkit/compiler

**鲲鹏GCC源码地址：**

https://gitee.com/src-openeuler/gcc

- **社区支持**

- **安全有保障：**鲲鹏GCC和开源GCC社区版本同步更新，同时针对安全问题做更为严格的分析和把控，及时合入CVE安全漏洞补丁；
- **开源、免费：**鲲鹏GCC免费提供发行包和开源代码，不收取任何费用；
- **技术支持：**鲲鹏GCC有专门的开发团队支持、维护，使用中的任何问题都可以通过社区和开发团队联系，并可以获得技术支持。