RVV 编译选项对性能影响的探究

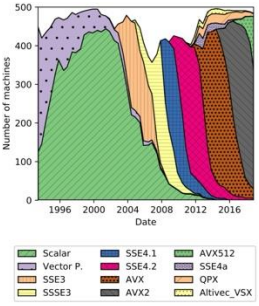
徐凯亮 黄知柏
上海交通大学 tcloud 实验室

RISC-V 中国峰会 2024

RVV 的重要性

- RVV：RISC-V Vector Extension，变长向量
- 备受关注的 RISC-V 高性能之路
  - RVA23 (v0.5: Candidate for Freeze Milestone Vote) 的必选
  - 后续高性能扩展（Zvb*, Zvk*）的依赖
- 工具链、配套环境跟进支持

现在 RVV 用起来性能怎么样了?



超算中向量指令集的使用

Evaluating Auto-Vectorizing Compilers through Objective Withdrawal of Useful Information.
DOI: 10.1145/3356842.

RVV 编译

汇编

```
saxpy:
    vsetvli a4, a0, e32, m8, ta, ma vle32.v
    v0, (a1)
    sub a0, a0, a4 slli a4,
    a4, 2 add a1, a1, a4
    vle32.v v8, (a2)
    vfmacc.vf v8, fa0, v0
    vse32.v v8, (a2)
    add  a2,  a2,  a4
    bnez a0, saxpy ret
```

编译器 intrinsic

```
void saxpy(size_t n, const float a, const float *x, float *y) {for (size_t vl; n > 0; n -
  = vl, x += vl, y += vl) {
    vl = __riscv_vsetvl_e32m8(n);
    vfloat32m8_t vx = __riscv_vle32_v_f32m8(x, vl); vfloat32m8_t
    vy = __riscv_vle32_v_f32m8(y, vl);
    __riscv_vse32_v_f32m8(y,__riscv_vfmacc_vf_f32m8(vy, a, vx, vl), vl);
} }
```

自动向量化

```
void saxpy(size_t n, const float a, const float *x, float *y)
{ for (size_t i = 0; i < n; i++) y[i] = a * x[i] + y[i]; }
```

简单的编程方式需要编译器支持

编译器自动向量化能力

实验编译器		GCC 14.1.0	Clang/LLVM 18.1.6
支持情况		(2023) GCC 13: RVV intrinsic (2024) GCC 14: RVV 自动向量化	借用 ARM SVE 等 (2022) LLVM 14 开始陆续支持
相关选项	优化等级	-O2/-O3/-Ofast	-O2/-O3/-Ofast
	目标架构	-march=rv64gcv[_zvlNNNb]	-march=rv64gcv[_zvlNNNb]
	向量化	-ftree-vectorize	-fvectorize
	LMUL	-mrvv-max-lmul={dynamic,m1,m2,m4,m8}	-mllvm -riscv-v-register-bit-width-lmul=LMUL
	向量长度	-mrvv-vector-bits={scalable,zvl}	-mrvv-vector-bits={scalable,zvl,VLEN}
			-mllvm -riscv-v-vector-bits-{min,max}=VLEN

编译器已经有功能支持，那么编译的 RVV 程序性能如何?

参考加速比例：用 intrinsic 的加速比

应用 llama.cpp，RVV intrinsic 加速计算 vs 标量计算 (-march=rv64gc[v])

设备 香蕉派 BPI-F3，CPU 进迭时空 SpacemiT K1（后同）

负载 Mamba 2.8b 模型（4bit 量化）

达成加速比

	模型加载 (s)	每秒处理提示词	每秒推理	总时间 (s)
标量	16.72	0.52	0.46	148.16
向量	9.60	2.95	1.77	738.95
加速比	1.74x	5.67x	3.85x	4.99x

intrinsic 可以取得很好的加速比，那自动向量化呢?

对比成熟向量：micro benchmark

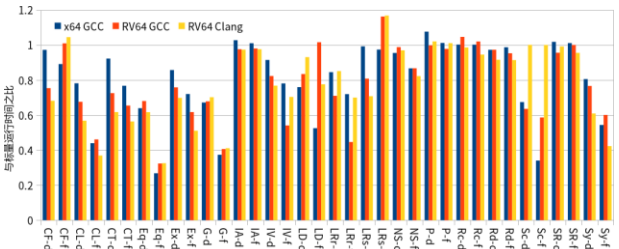
测试套件 TSVC

向量加速比例（几何平均）

x64 23%

RV64 GCC 24%

RV64 Clang 25%

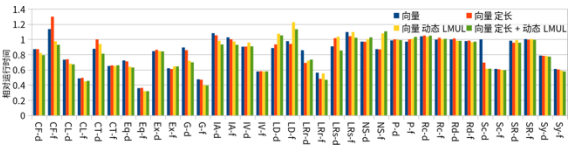


micro benchmark 上可以达到成熟架构水平

RVV 自动向量化：RVV 特有选项

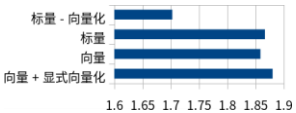
- 向量定长（VLS）vs 变长（VLA）
  VLS 比 VLA 再加速 1%
- 最大 LMUL 设置
  设置动态 LMUL 再加速 2%
  (设置 LMUL=2 可加速 5%)

- 目前 VLS 编译缺乏优势
- 性能调优需要考虑合适的 LMUL

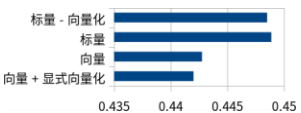


GCC 自动向量化（O3 优化）
默认 LMUL=1 变长向量，比标量加速 19%（几何平均）

对比成熟向量：application benchmark



x64@4.8GHz x 8 核
CPU2017speed/GHz



RV64@1.6GHz x 4 核
CPU2017speed/GHz

测试 CPU2017speed 中的 C/C++ 程序，去除会导致 OOM 的任务

在部分应用程序上，RVV 自动向量化是负收益

- 自动向量化不适合对全应用开启
- 原因：编译器的代价模型与实际不匹配
- 其它原因?

结论?

- VLS 相比 VLA，有性能优势，但优势不大
- RVV 自动向量化对宏观性能收益不明显，还不适合对全应用开启
- RVV 自动向量化与成熟向量性能收益有 gap
- 有待根据硬件体系架构的编译调优

实验局限性

限于硬件可用性，实验涉及硬件种类单一，影响结论的普适性。
编译器工具链在不断发展，性能效果会不断变化。

谢谢

徐凯亮 黄知柏 · RVV 编译选项对性能影响的探究