开放原子开源基金会 OPENATOM FOUNDATION | OpenEuler

全流程预取优化

演讲人：黄晓权

演讲人职位：华为-高级工程师

目录

1. 背景及问题
2. 技术方案：全流程预取优化
    编译器预取优化
    BOLT二进制预取
    运行时预取调节
3. 应用效果
4. 未来展望

背景及问题

内存瓶颈加剧、软件预取困难的原因是什么?

硬件方面

- **核心数持续增多，而每核平均的LLC容量有限**
- 不同型号和代次，**硬件预取策略不同**
- **访存带宽的压力**和**多核并行预取，**CPU频率变化及指令乱序执行

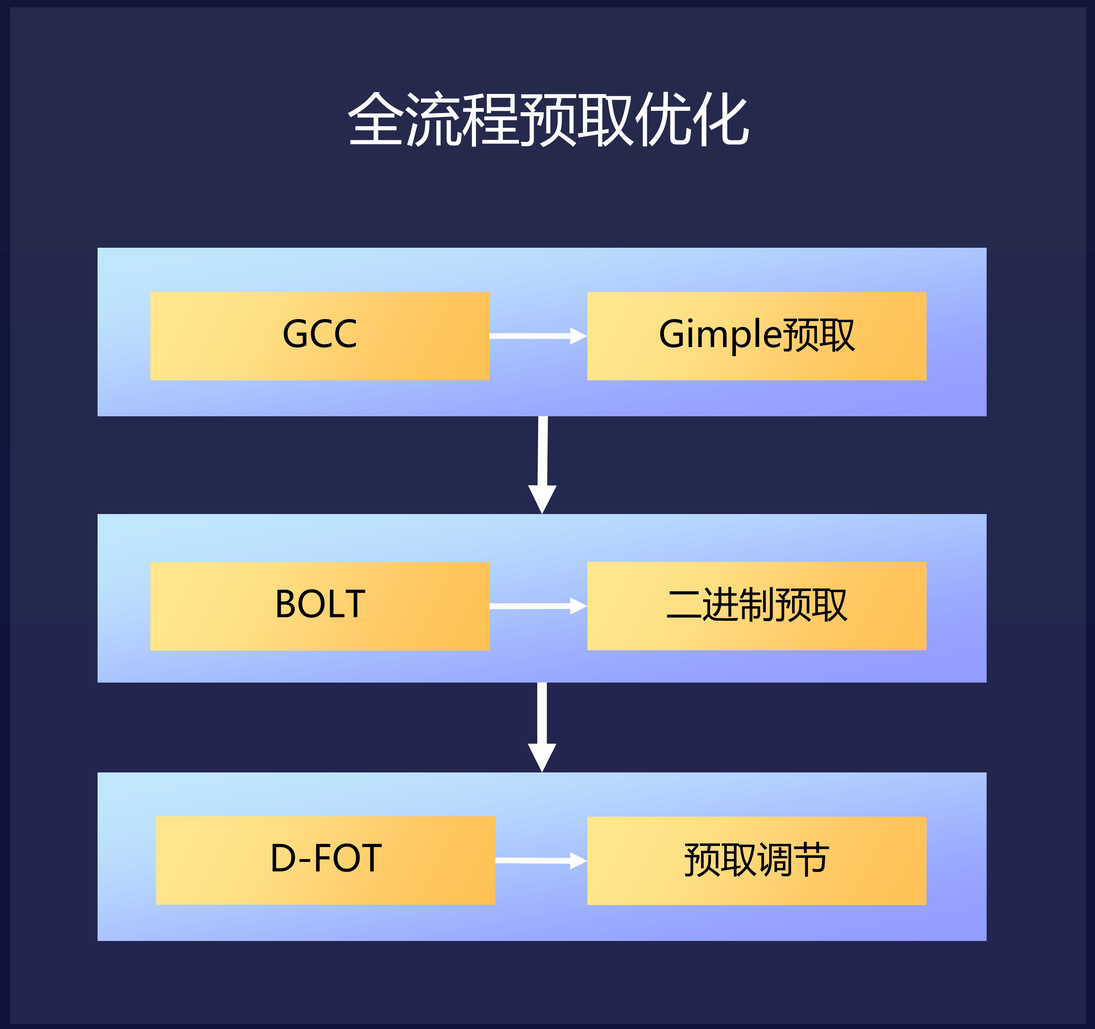
软件方面

- 大数据、数据库、搜推召回等**软件规模不断扩大**，数据访问量持续增加
- 基于人工分析或静态编译，**无法解决复杂场景的预取**（如间接访存，指针访问）
- **不恰当的预取或过多预取，**都可能带来负收益

解决方案

- 综合**静态、反馈、运行时**的预取能力，构建一套**适应性强的全流程预取方案**，充分适应复杂场景及硬件差异

2. 总体方案：全流程预取优化



GCC 中：基于CFG图**增强分支加权的预取计算**，但依赖源码编译

BOLT中：**新增二进制预取**pass，结合profile和度指令片段**可适用并补充更多插入场景**，但难以准确计算预取值

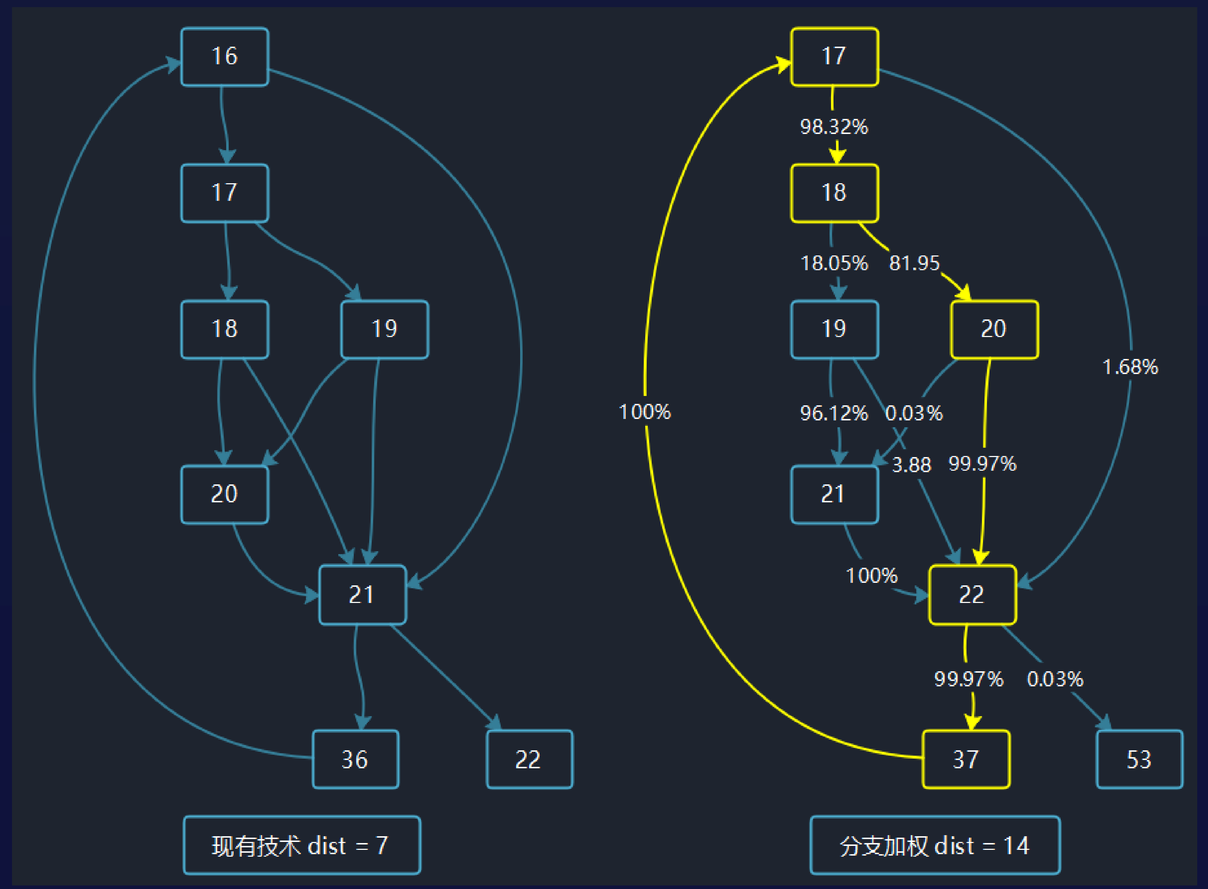
D-FOT：**新增运行时预取调节策略**，对插入的预取指令，进行**函数替换和预取值修改**，根据硬件特性调节最佳预取

优化环节	优势	不足
GCC gimple预取	分析灵活	依赖源码
BOLT二进制预取	适用更广	计算复杂
D-FOT运行时调节	灵活调节	-

2.1 编译器预取优化增强

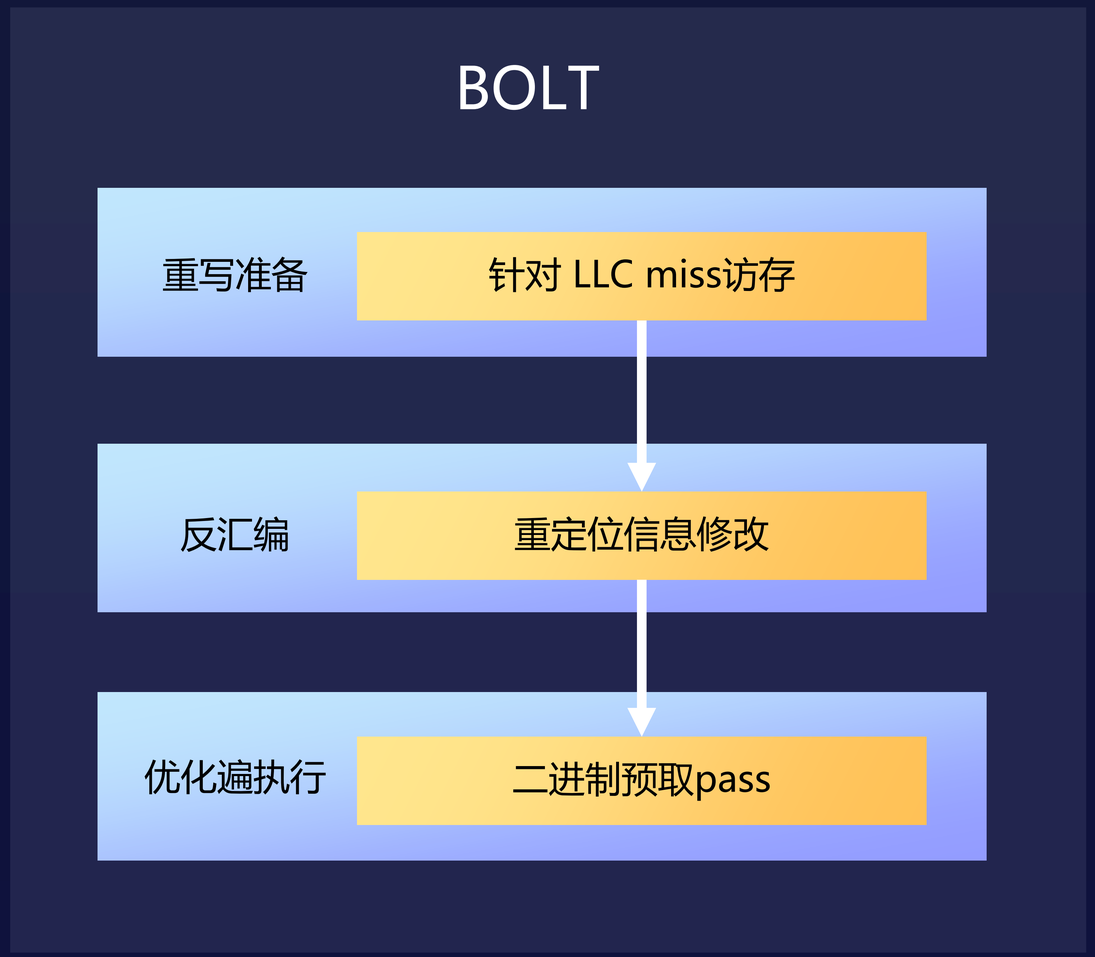
```
for (const BinaryBasicBlock *BB : BF.layout()) {
  for (const MCInst &Inst : *BB) {
    // Tail calls are marked as implicitly using the stack
    // could be inlined.
    if (BC.MIB->isTailCall(Inst))
      break;

    if (BC.MIB->isCFI(Inst)) {
      HasCFI = true;
      continue;
    }

    if (BC.MIB->isCall(Inst))
      IsLeaf = false;

    // Push/pop instructions are straightforward to handle.
    if (BC.MIB->isPush(Inst) || BC.MIB->isPop(Inst))
      continue;

    DirectSP |= BC.MIB->hasDefOfPhysReg(Inst, SPReg) ||
                BC.MIB->hasUseOfPhysReg(Inst, SPReg);
  }
}
```

循环中存在**多种分支跳转**，预取值无法有效计算



结合反馈标注CFG边概率，**加权计算循环中主要执行路径**，获得更准确的预取距离

2.2 BOLT二进制预取



问题：当前BOLT**没有预取**优化的能力

动机：在无源码情况下，**对二进制进行预取插入**

方案：基于BOLT框架，**新增一个二进制的预取**PASS

- 重写准备中，**针对Miss率高的访存进行筛选分析**
- 反汇编函数中，增加**重定位的信息创建**，对优化后的函数地址，进行指向使用

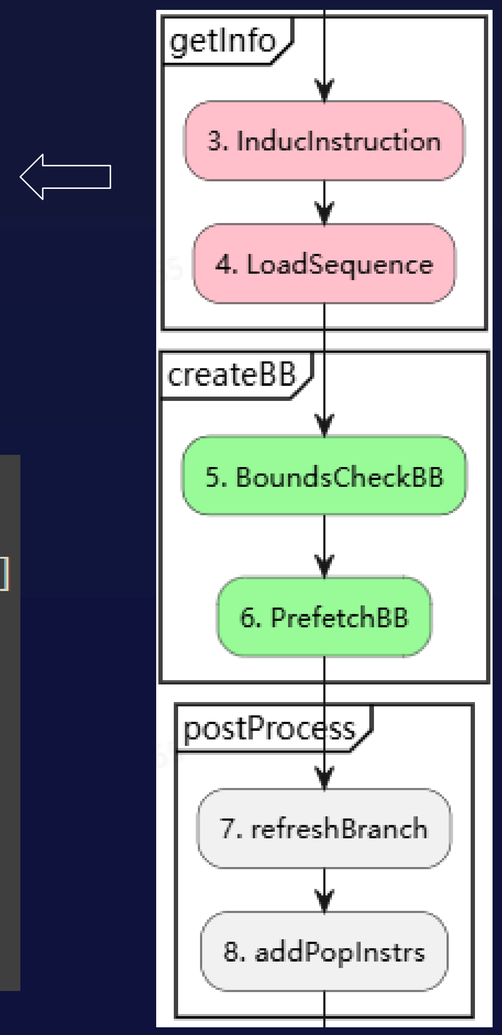
延伸：二进制预取可**为运行时调节提供准备**

2.2 BOLT二进制预取

获取汇编循环信息：

1. 获取循环的**归因变量和判断指令**
2. 获取**热点访存序列**的指令片段
3. 根据核心片段分析寄存器分配信息

```
归因变量: add     x9, x9, #0x8
边界判断: cmp     x9, #0x400
LLCMiss:  str     x10, [x25, x9, lsl #3]

访存指令序列:
ldr     x10, [x25, x9, lsl #3]
asr     x11, x10, #63
and     x11, x11, #0x7
eor     x10, x11, x10, lsl
and     x11, x10, x24
lsl     x11, x11, #3
ldr     x12, [x19, x11, lsl #3]
```



创建预取片段：

1. 根据访存片段和寄存器映射表，**新生成访存指令片段**
2. 创建**边界检测的片段**，支持间接预取
3. 插入预取指令，创建跳转分支组合预取片段

```
mov     x9, x2
add     x2, x2, #0x100
cmp     x2, #0x400
b       800154 <main+0x154>
ldr     x0, [x25, x9]
asr     x1, x0, #63
and     x1, x1, #0x7
eor     x0, x1, x0, lsl #1
and     x1, x0, x24
lsl     x1, x1, #3
ldr     x2, [x19, x1]
prfm    pldl1keep, [x25, x2]
```

2.3 运行时预取调节

二进制的预取中，预取距离计算较为困难

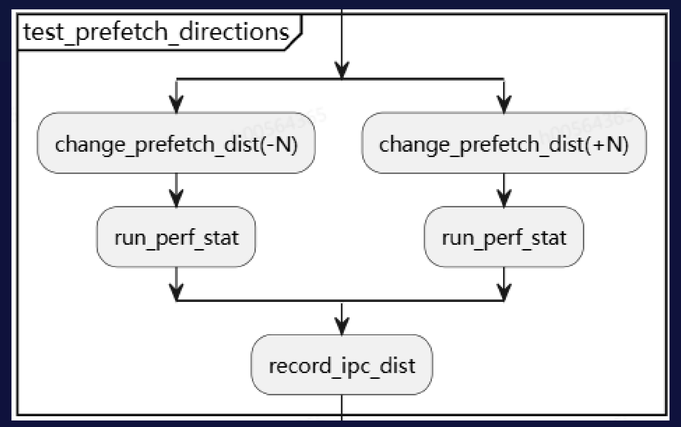
通过运行时调节，能结合硬件特征，调节最优值

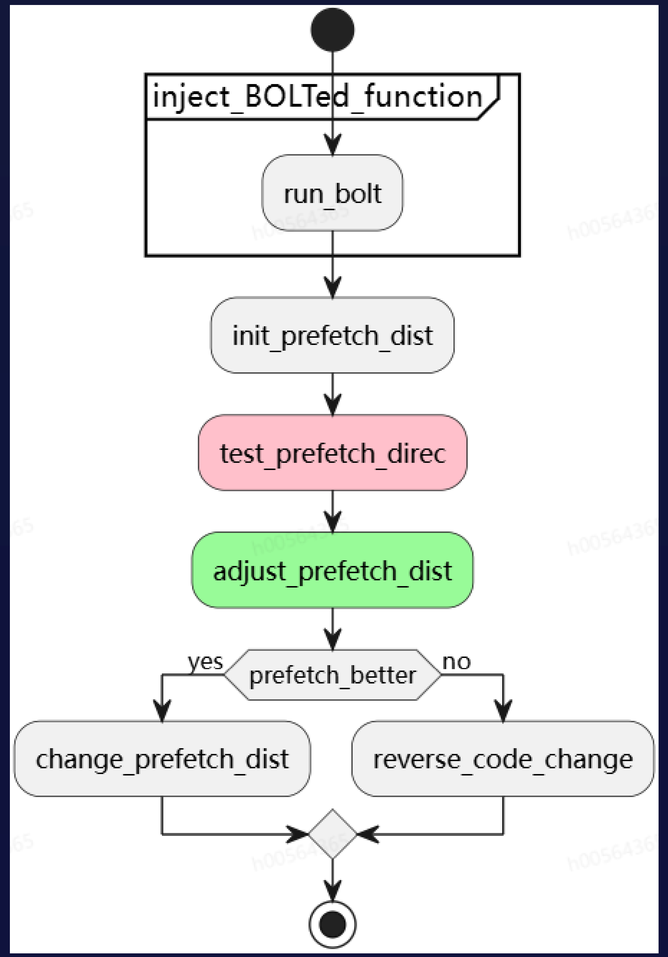
- 通过D-FOT，**启动创建子进程，调用目标程序和预取修改插件**
- 运行时中，通过**实时采集IPC分析预取值效果**，并调用预取调节策略进行最优调节

2.3 运行时预取调节

预取方向分析：

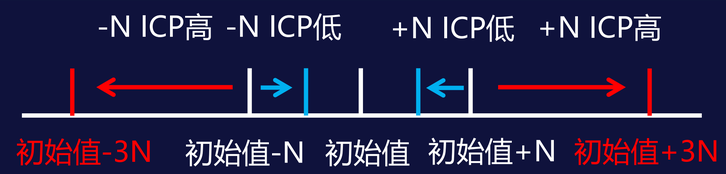
- 运行时修改预取距离值，分别调节：**初始值−N，初始值，初始值+N**；
- 分别采集不同预取值的IPC情况，**对比与初始值的IPC**，判断是否往-N/+N进行调节。



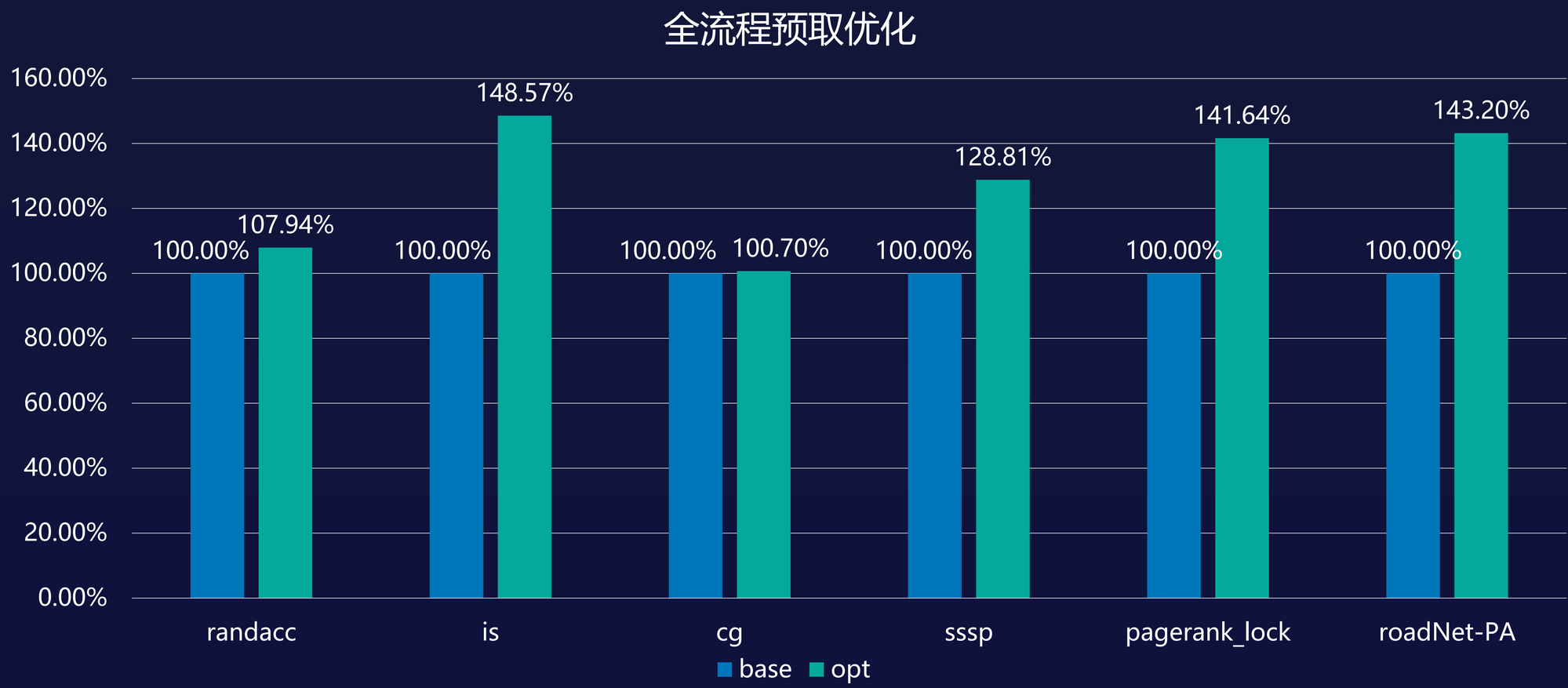


预取距离调节计算：

- 如果±N的IPC提高，即继续**二倍扩展**
- 如果±N的IPC降低，即在±N和中间值**之间二分**，直到调节到最优值
- 快速迭代查找出最优的预取值



3. 应用效果



全流程预取优化，在多个benchmark上的初步验证，有**约7%~40%**的性能提升（平均约有20%+），但对于部分无优化效果的场景，也可以通过回退预取优化，避免负优化和影响。

4. 未来展望

预取及内存优化展望：

- 二进制预取结合运行时调节，主要针对预取距离，未来也可考虑预取插入位置可调
- 运行时调节结合的反馈是IPC，如果能结合软件插桩，可更准确进行效果反馈
- 可编程的软硬协同预取和分配，可以与当前预取识别和插入再深度结合
- 结合动态软件信息，在对象冷热识别和堆的分配和排布上，可针对场景进行布局调优，提高空间局部性