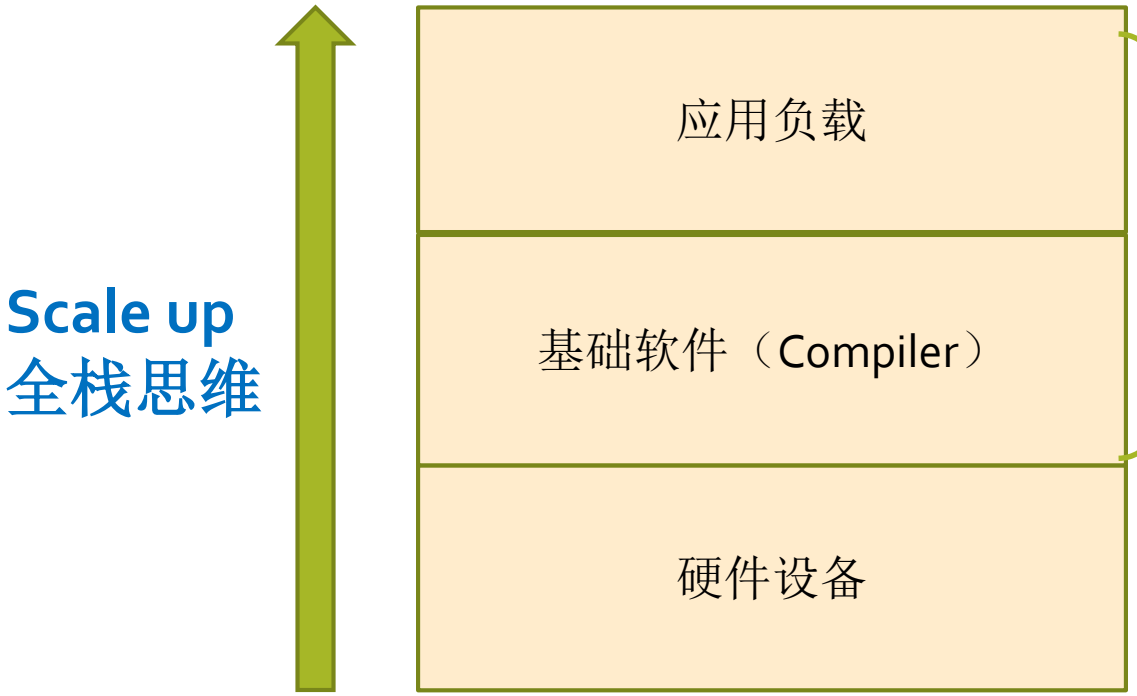
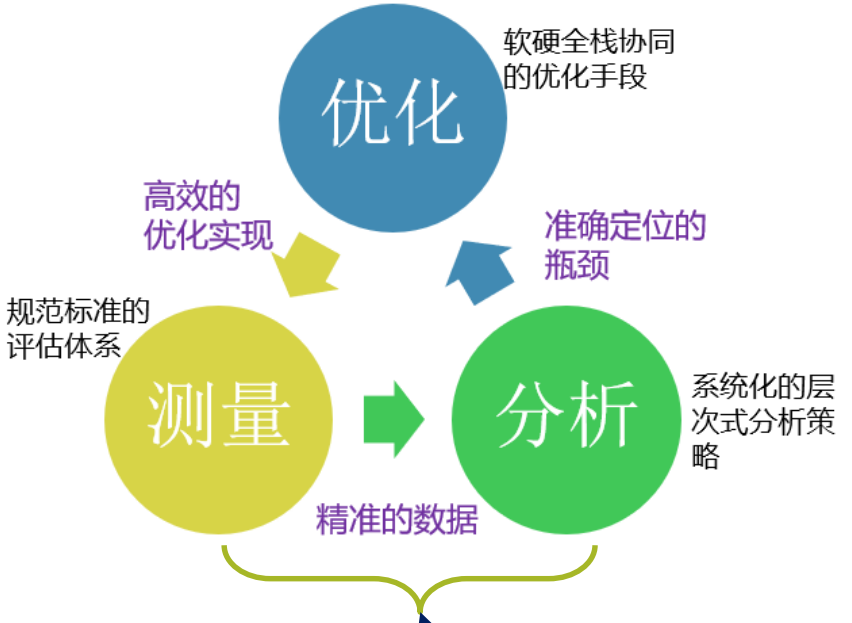
程序插桩及优化机会识别

黄 波
bhuang@dase.ecnu.edu.cn

SOLE 系统优化实验室
华东师范大学

本次课的关注点



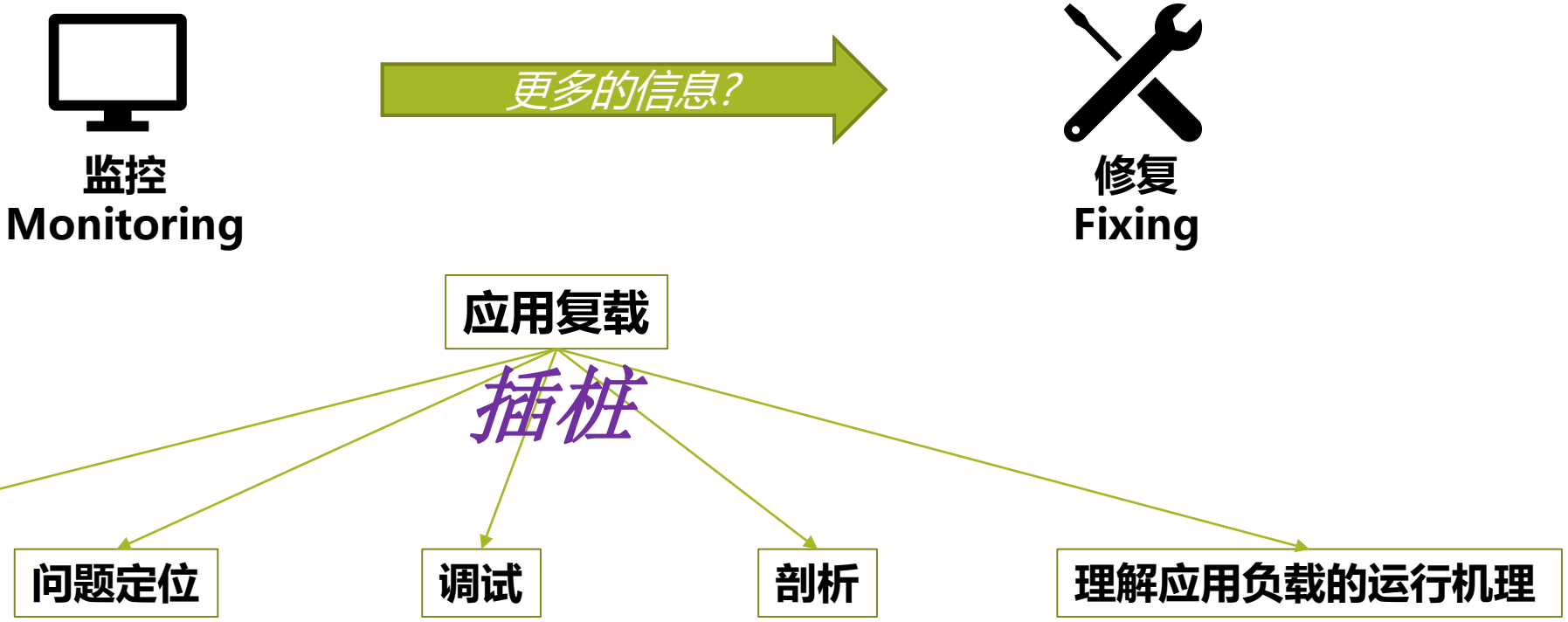


- 理解应用负载的动态特征及行为是程序优化的前提和基础
- 插桩（Instrumentation）是理解应用负载动态特征的一种常见手段
- 插桩工具的实现通常需要用到程序分析及编译技术
- 插桩工具可以用来帮忙寻找优化机会

本次课程内容

- 什么是程序插桩?
- 二进制翻译如何助力程序插桩?
- 如何利用插桩信息识别编译优化机会?

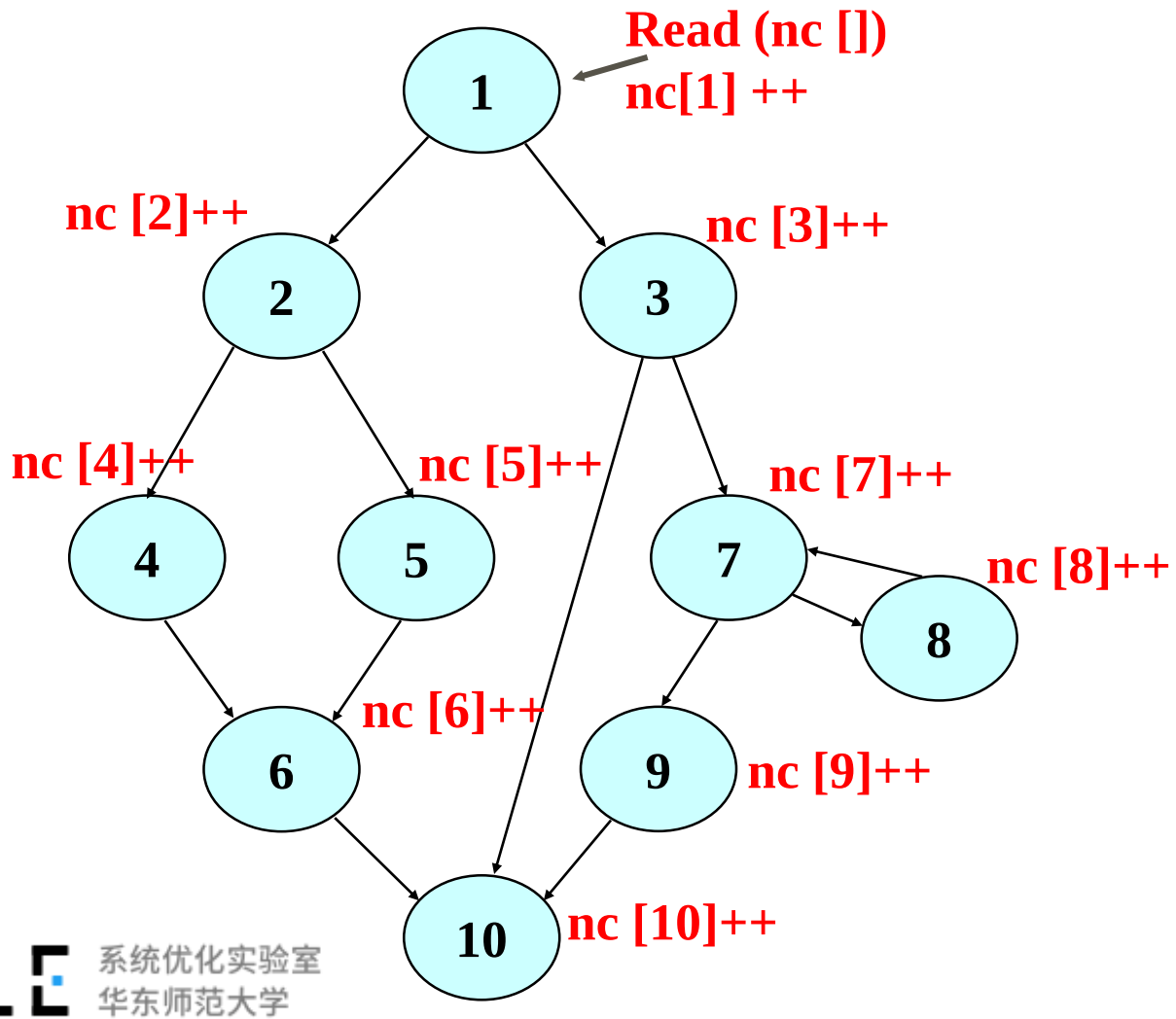
为什么需要进行程序插桩?



什么是插桩?

- 从根本上来说，插桩是从一个应用负载中获得一些特别关注的测量结果的过程；
- 在实践中，插桩也可以是非常简单的测量操作，譬如记录每个函数的执行时间以方便定位性能瓶颈的发生点；
- 严格来说，程序插桩是指在被插桩程序中插入一条或者多条语句/指令以获取程序运行时的某些信息或者特征，比如：
  - 完成诸如典型输入情况下程序运行时的行为特征收集、代码执行时间测量、日志记录、执行路径获取等
- 插入的语句不能影响被插桩程序的行为，也应该尽量减少对被插桩程序的性能影响
- 根据被插桩程序的文件格式，程序插桩包括源代码级别的程序插桩和二进制代码级别的程序插桩

程序中语句覆盖的插桩示例

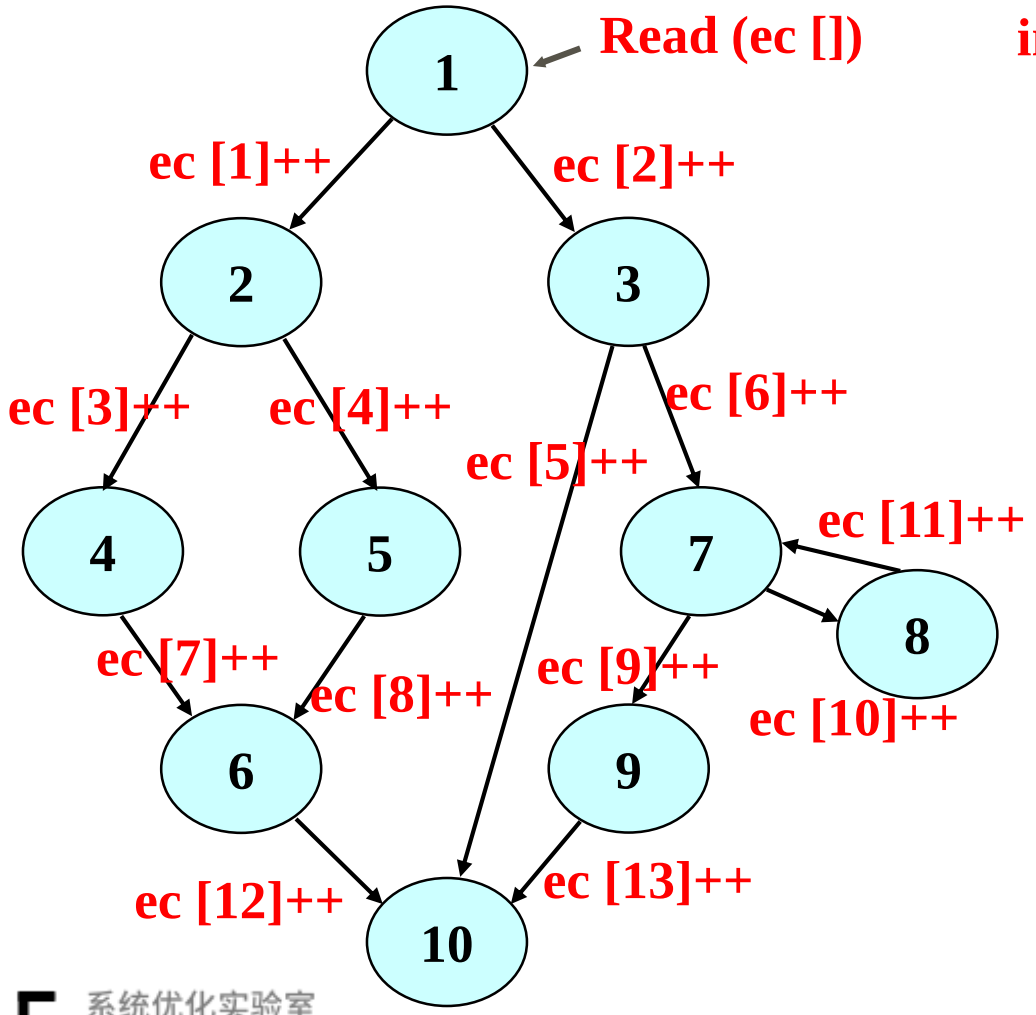


int nc[] = {0,0,0,0,0,0,0,0,0,0,0}

每一个节点代表一个基本块

如果运行完一系列的测试后，某个基本块（假设为node）的执行次数为零（即*nc[node]*的值为零），则可以得出此基本块没有被覆盖的结论。如果某些基本块的执行次数很大，说明这些基本块是潜在的优化热点

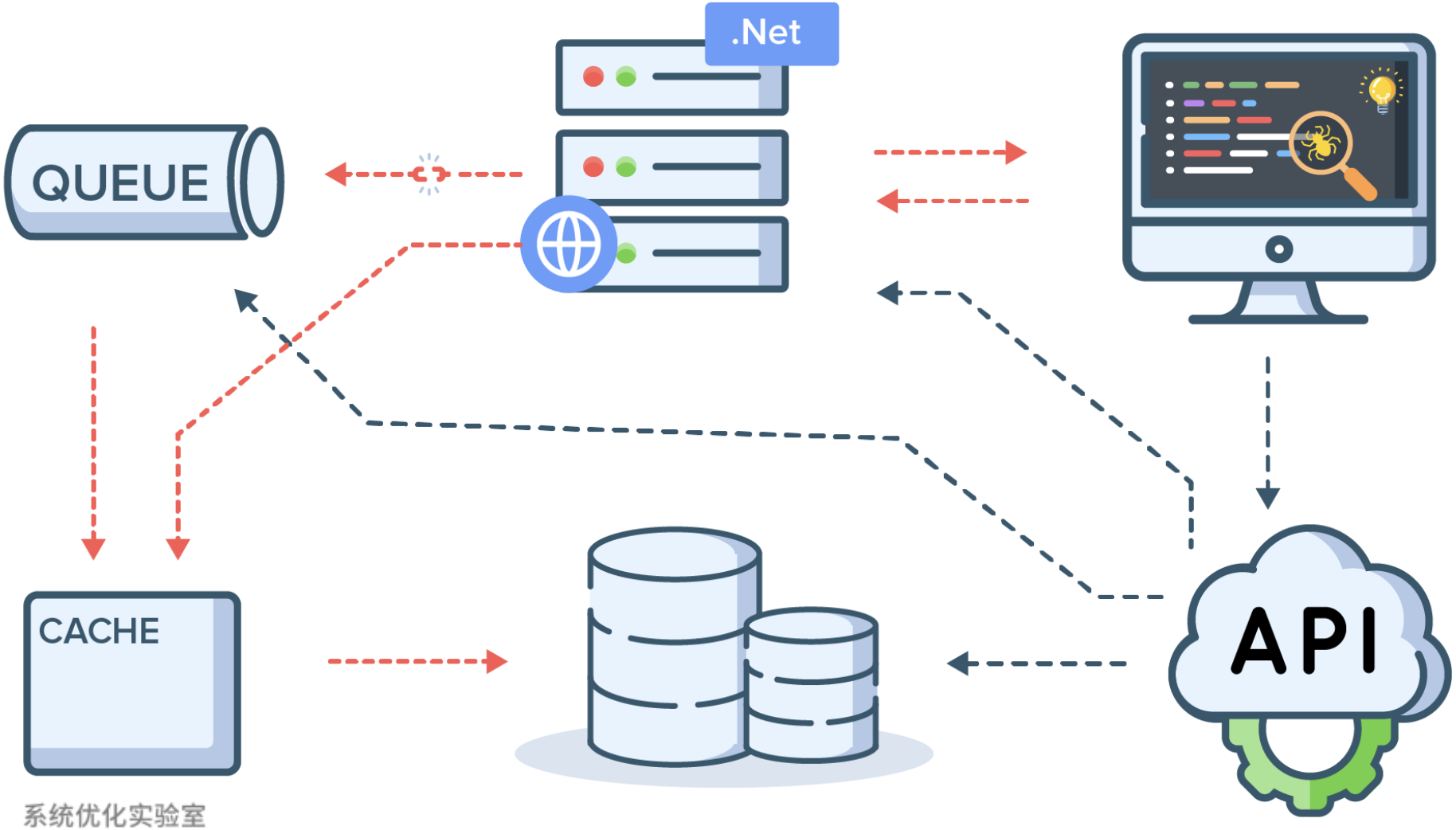
程序中边覆盖的插桩示例



int ec[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}

- 如果某条边e对应的执行次数为零（即*ec[e]*的值为零），则说明e没有被覆盖；
- 如果某些边的执行次数很大，说明这些边对应的控制流跳转频繁发生，这些信息对形成程序执行时的热点路径（hot trace）非常重要；
- 如果ec变成一个布尔数组，则ec中每个数组元素记录的便是对应的边是否被覆盖的信息。

分布式执行路径跟踪（Distributed Tracing）



程序插桩的手段

- 人工插桩（Manual）
- 源代码级别的自动插桩（Automatic Source Level）
- 中间语言级别的插桩（Intermediate Language）
- 编译器助力插桩（Compiler Assisted）
- 静态二进制重写（Static Binary Rewriting）

静态插桩

- 运行时插桩（Runtime Instrumentation）
- 运行时注入（Runtime Injection）

动态插桩

https://en.wikipedia.org/wiki/Profiling_(computer_programming)

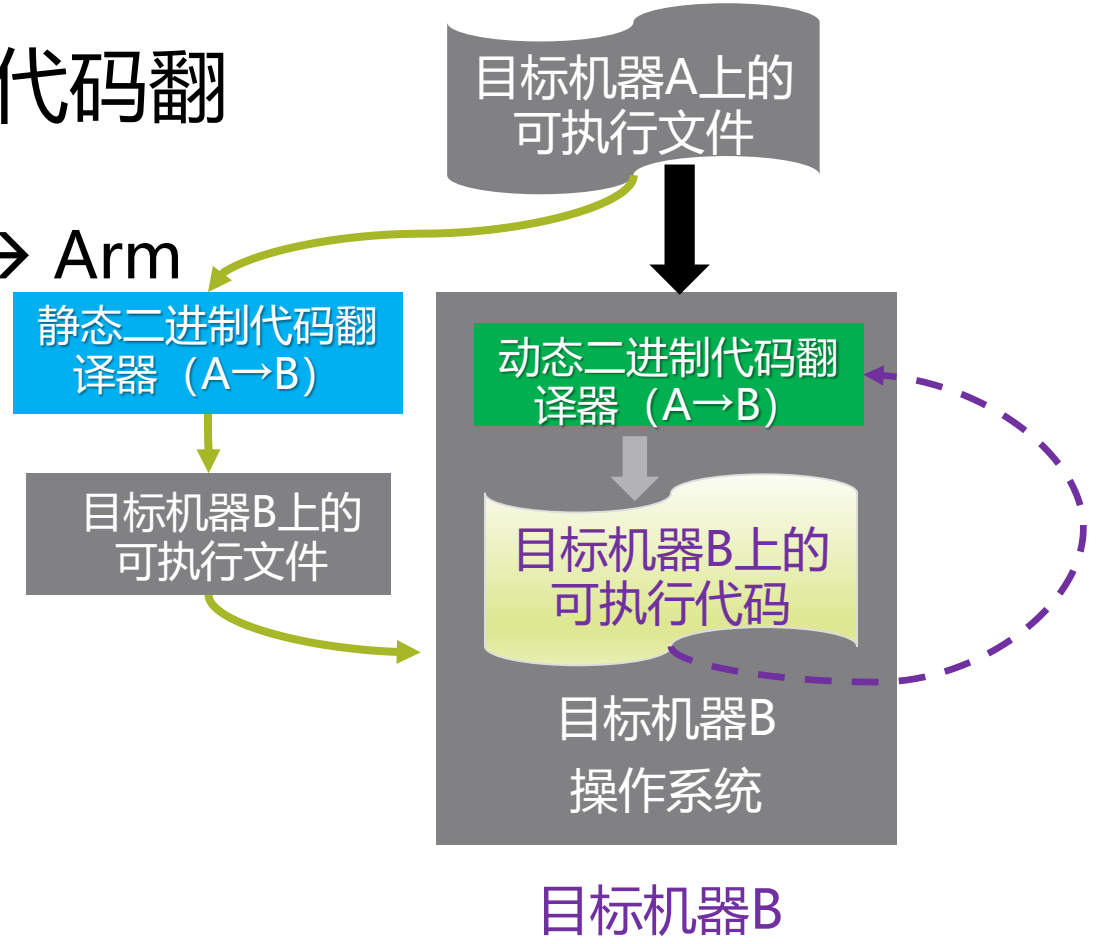
静态插桩 vs. 动态插桩

静态插桩	动态插桩
● 离线插桩，无需过多考虑被插桩程序的解析时间及插桩代码插入的时间； ● 可能产生更加有效的插桩后代码； ● 无法对运行时的事件进行立即响应；	● 在程序运行的特定时间内进行插桩，时间和开销敏感； ● 在程序运行过程中插入/移除插桩代码； ● 对运行时的事件可以第一时间及时响应；

什么是二进制翻译(Binary Translation)?

- 二进制翻译的功能是把一种二进制代码翻译成另外一种二进制代码:
  - 可以是跨指令集架构的翻译，如X86-64 → Arm
  - 也可以是同指令集架构之间的翻译

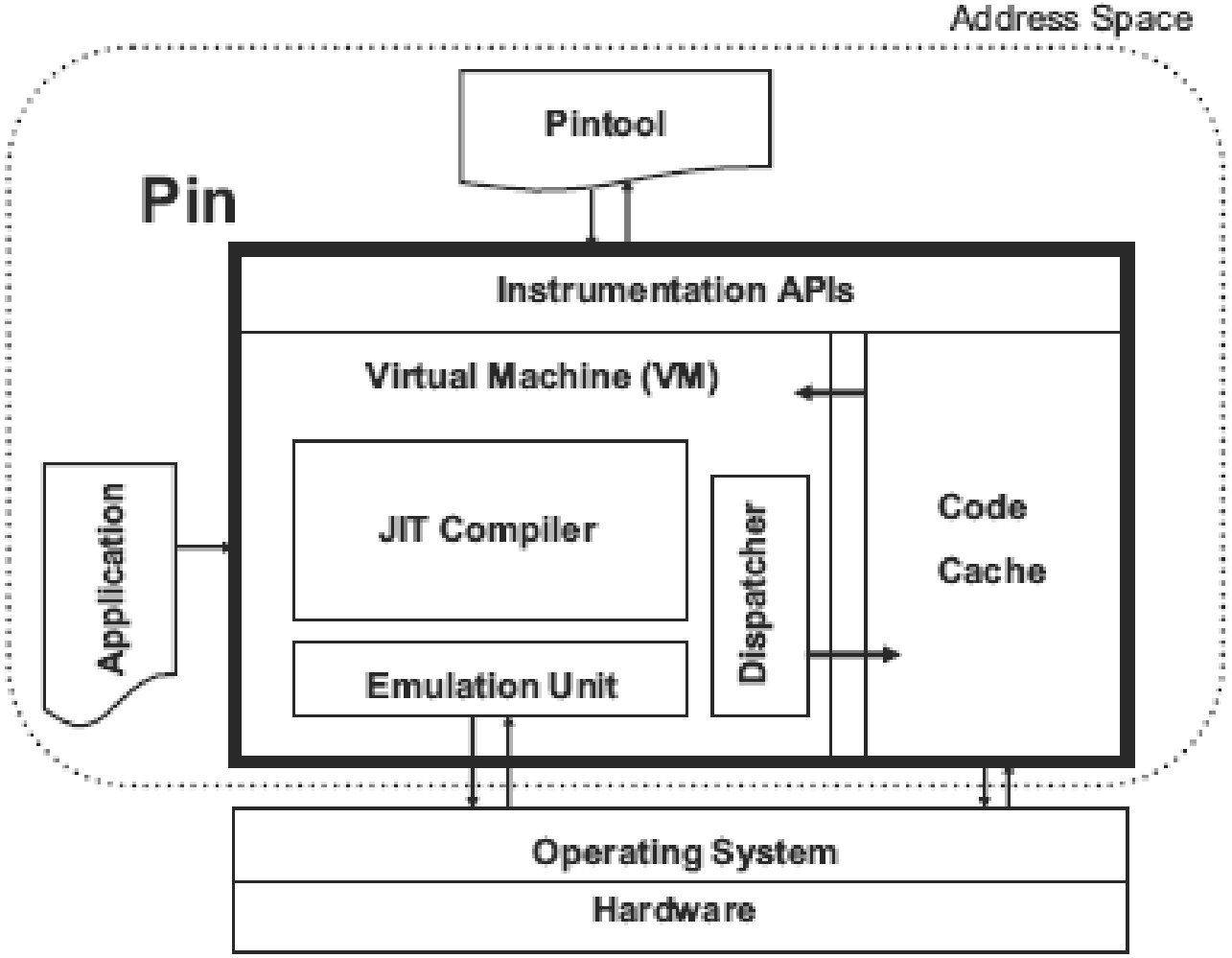
- 同指令集架构之间的二进制翻译是插桩工具开发时常用的技术:
  - 静态二进制重写
  - 运行时插桩



Pin的软件系统架构

https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html

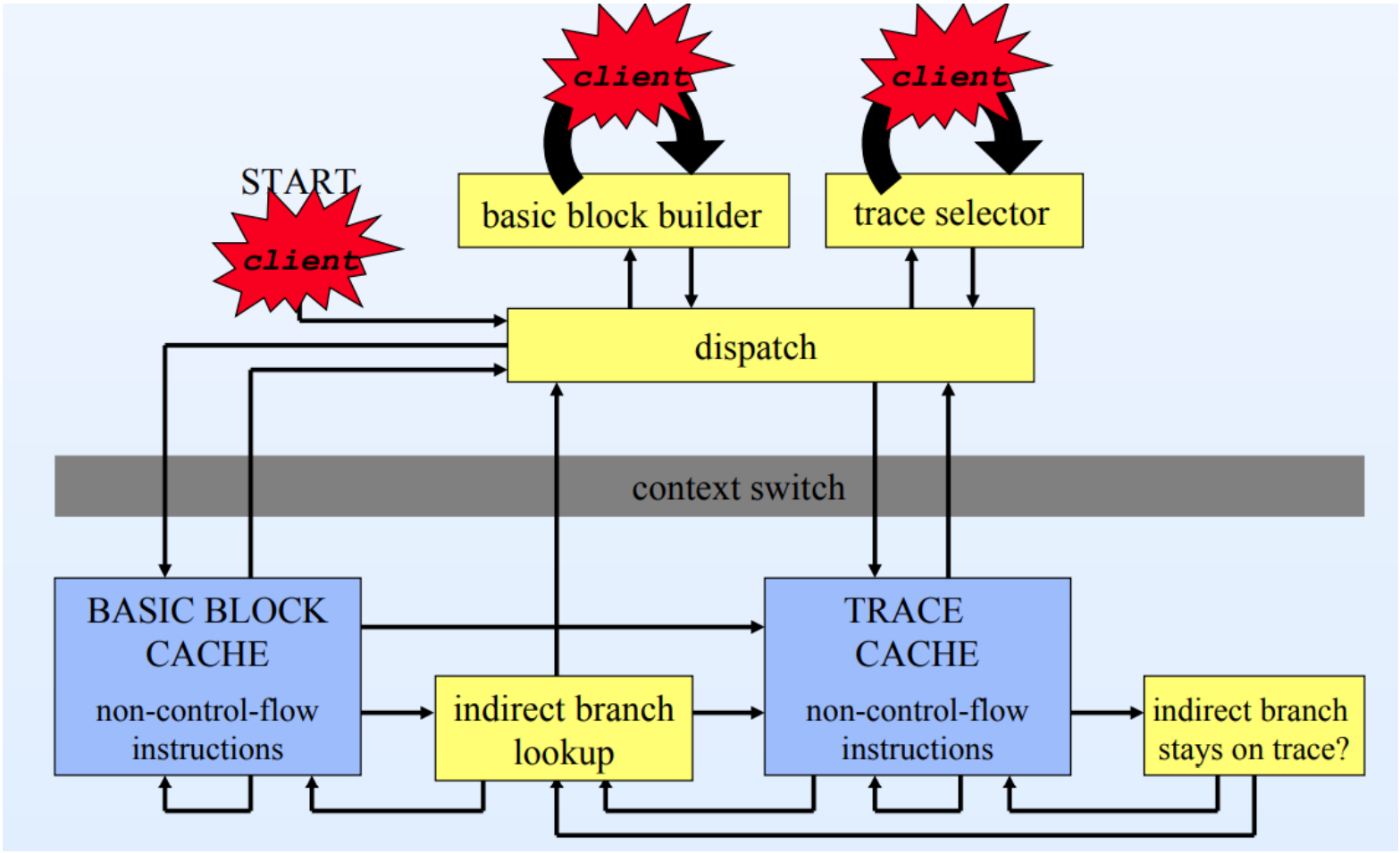
免费但不开源



只支持在Intel平台上的插桩及动态优化!

Dynamo-RIO系统架构

https://dynamorio.org/



DynamoRIO是开源的，支持多平台

编译优化机会识别：源码分析

```
do i = 1, n
  a[i] = a[i] + c*i
end do
```

(a)

```
T = c
do i = 1, n
   a[i] = a[i] + T
   T = T + c
end do
```

(b)

强度削弱

```
do i = 2, n
   a[i] = a[i] + c
   if (x < 7) then
      b[i] = a[i] * c[i]
   else
      b[i] = a[i-1] * b[i-1]
   endif
end do
```

(c)

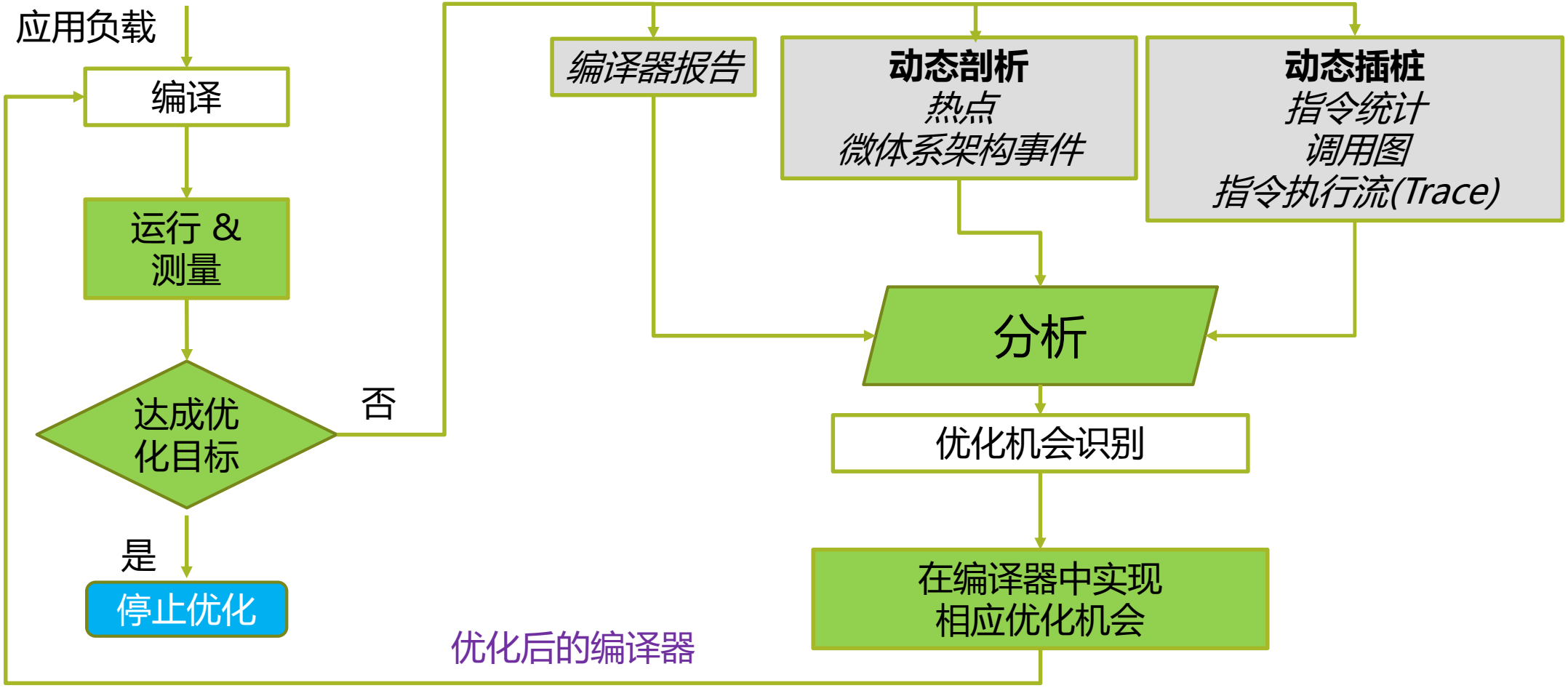
```
if (x < 7) then
   do all i = 2, n
      a[i] = a[i] + c
      b[i] = a[i] * c[i]
   endi do all
else
   do i = 2, n
      a[i] = a[i] + c
      b[i] = a[i-1] * c[i-1]
   endi do
end if
```

(d)

Unswitching

早期的优化机会识别通常针对循环来展开

编译优化机会识别：常用方法



高度依赖于从事编译优化分析工作人员的经验及专业程度

不同编译器的优化能力有差异

- 开源编译器

- 商业编译器产品

毕昇编译器

Intel® oneAPI DPC++/C++ Compiler
A Standards-Based, Cross-architecture Compiler

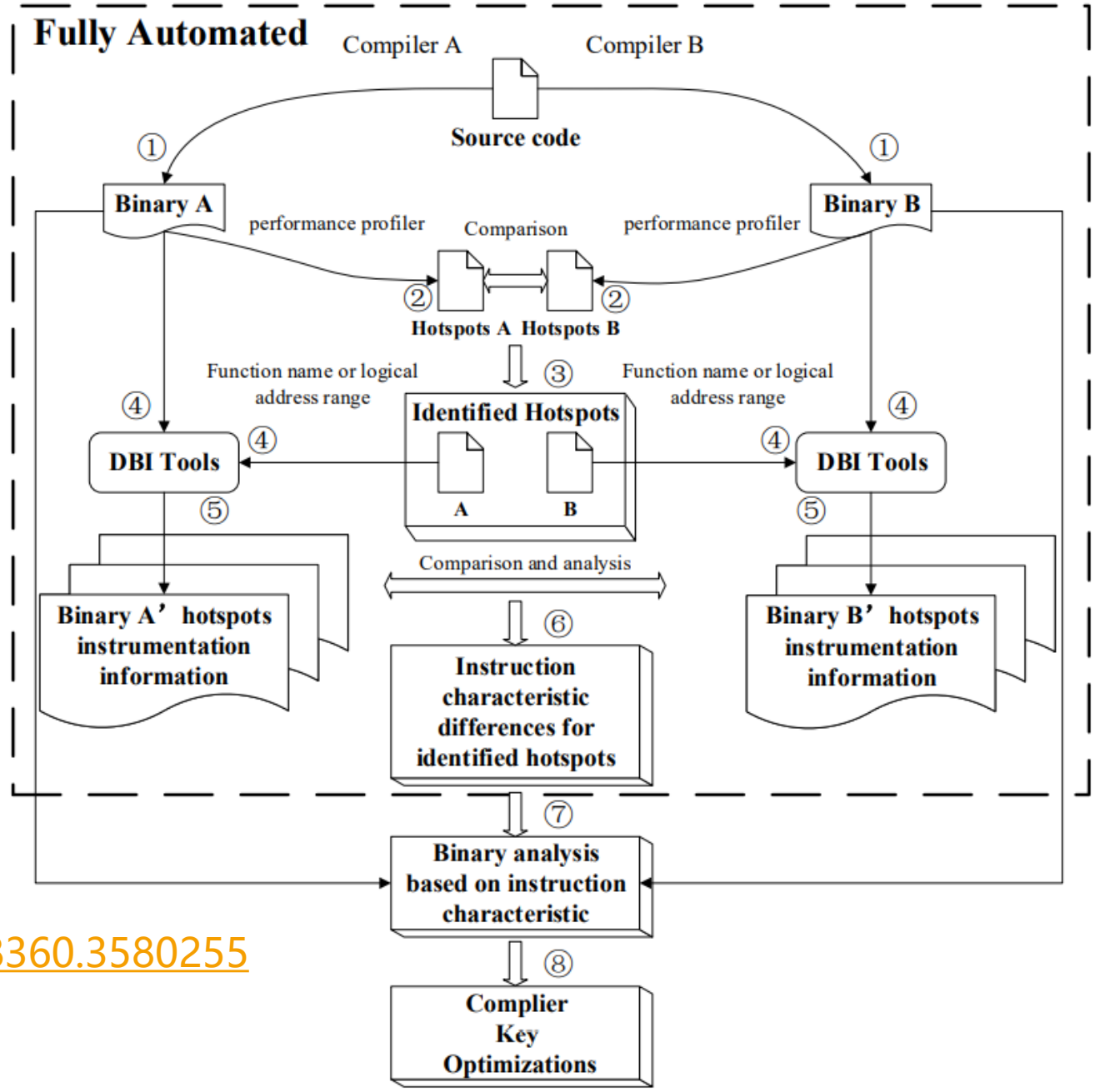
AMD Optimizing C/C++ and Fortran Compilers (AOCC)

Q: 是否可以"学习"别的编译器的优化方法?

一种新方法

一个热点驱动的半自动化编译器竞争分析框架

- ①-⑥是可以自动完成的，⑦和⑧是必须人工介入的
- ①-⑥的自动化大大节省了数据收集及分析的时间，也大大缩小了后期人工分析的程序范围



https://dl.acm.org/doi/10.1145/3578360.3580255

插桩工具对X64和AArch64的指令分类

Instruction Category			X64	AArch64
Branch	Indirect	Jump	jmp (mem,reg)	br
		Call	call(mem,reg)	blr
		Ret	ret	ret
	Conditional		jle/jne/je/ja/jna …	b.cond/cbz/cbnz/tbz/tbnz
	Unconditional Direct		jmp/call imm	b/bl
Operation	Binary Arithmetic		cmp/sub/mul/vaddps…	cmp/sub/mul/add…
	Logical		and/or/xor/test/vandps…	add/or…
	Shift		shr/sha/sar/sal	asrv/rorv/lslv/bfm…
Data Transfer			mov/movzx/vmovd…	movi/movz/fmov…
Stack			push/pop	——
Load/Store			——	str/stp/ldr/ldp…
Others			lea/setb/nop…	adrp/adr/nop/csel…
Vectorization Instruction			SSE/AVX/AVX2/AVX512	Neon/SVE

实验配置

	X64	AArch64
Processor	Intel®Xeon®Gold 5218R	Huawei Kunpeng 920-5250
CPUs	80 (logical CPUs, 2 sockets, Hyperthreading ON)	96 (physical cores, 2 sockets)
Open source compiler	GCC (version 10.3.0)	GCC (version 10.3.0)
Proprietary Compiler	Intel's ICC compiler (version 2021.6.0)	Huawei's BiSheng compiler (version 2.1.0)
Operation System Version	Ubuntu 20.04.4 LTS Linux	Ubuntu 20.04.4 LTS Linux

- 编译SPEC CPU® 2017所用的配置是从SPEC CPU® 2017的官方网站上找到的跟上表实验平台最接近的配置
- *runcpu*命令行的主要选项用的是*tune=base, action=validate* 以及 *size=train*

案例分析负载选择

Benchmark	Platform	GCC	BiSheng/ICC	Gap
Fortran				
648.exchange2_s	AArch64	38.9	25.0	55.6%
C++				
620.omnetpp_s	X64	45.8	31.2	46.8%
631.deepsjeng_s	X64	83.0	68.0	22.1%
641.leela_s	X64	89.2	76.0	17.4%
623.xalancbmk_s	X64	35.2	35.2	0.0%

编译器那两栏中的数据是对应行的基准测试程序用对应列的编译器编译出来可执行文件的运行时间（以秒计算）

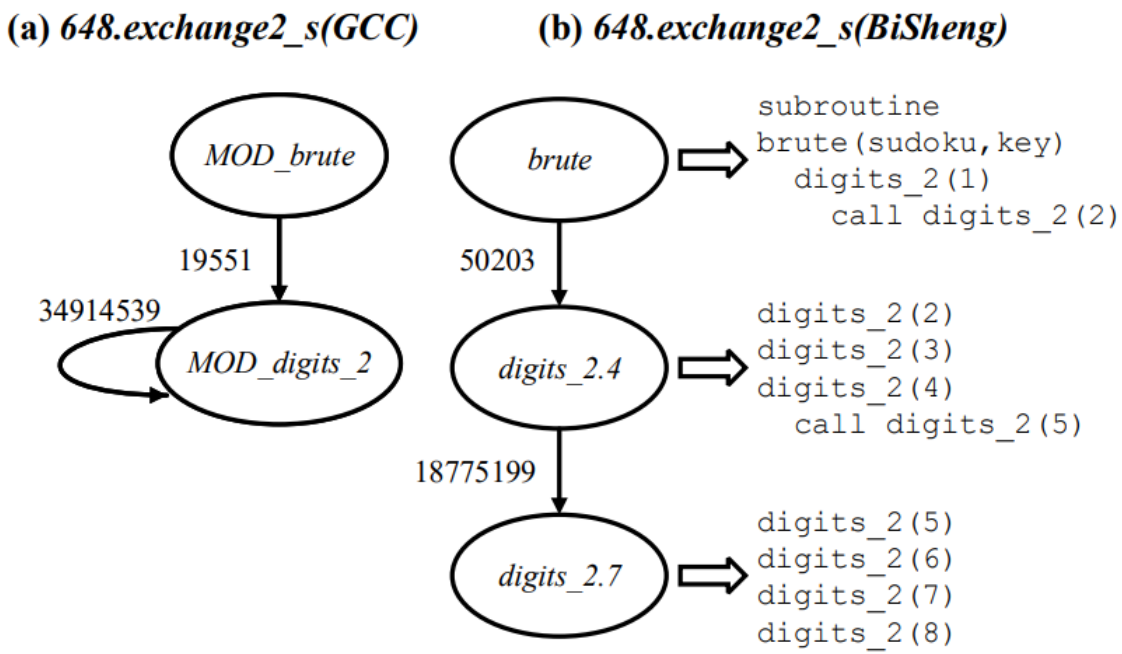
648.exchange2_s(GCC)和648.exchange2_s(BiSheng)的热点比较

Rank	BiSheng			GCC		
	relative time(%)	absolute time(sec)	function name	relative time(%)	absolute time(sec)	function name
1	**57.32**	14.33	*digits_2.7*	**82.38**	32.05	*MOD_digits_2*
2	**19.38**	4.85	*digits_2.4*	**7.30**	2.84	*gfortran_mminloc*
3	**18.30**	4.58	*logic_new_solver*	**4.15**	1.61	*specific.4*
4	1.46	0.37	*free*	**2.31**	0.90	*logic_MOD_new_solver*
5	0.80	0.20	*malloc*	1.08	0.42	*hidden_triplets.0*
6	0.43	0.11	*covered*	0.78	0.30	*free*
7	0.32	0.08	*brute*	0.55	0.21	*naked_triplets.1*
8	0.17	0.04	*f90_dealloc*	0.48	0.19	*hidden_pairs.2*
9	0.15	0.04	*f90_alloc*	0.25	0.10	*MOD_brute*
10	0.14	0.04	*f90_set_intrin*	0.25	0.10	*malloc*

嫌疑热点

编译优化机会识别（1）

Category	*digit_2.4*	*digit_2.7*	*MOD_digit_2*
Branch	3593142992	11609217450	29321533296
Indirect	50203	18775199	34934090
Jump	0	0	0
Call	0	0	0
Ret	50203	18775199	34934090
Cond	3361685141	10733730071	27698651020
Uncond Direct	231407648	856712180	1587948186
Operation	15199221930	41803072569	116664503804
Binary Arith	14686454611	39759170962	102813533806
Logical	498710685	1733736947	12559811188
Shift	14056634	310164660	1291158810
Data Transfer	32692266	214161627	1636029315
Load/Store	11911031090	37507748969	81757176763
Others	534241804	1514087575	2581871188
Vectorization	5426852903	10429222578	268662027



函数内联 & 函数特化

编译优化机会识别（2）

Category	*digit_2.4*	*digit_2.7*	*MOD_digit_2*
Branch	3593142992	11609217450	29321533296
Indirect	50203	18775199	34934090
Jump	0	0	0
Call	0	0	0
Ret	50203	18775199	34934090
Cond	3361685141	10733730071	27698651020
Uncond Direct	231407648	856712180	1587948186
Operation	15199221930	41803072569	116664503804
Binary Arith	14686454611	39759170962	102813533806
Logical	498710685	1733736947	12559811188
Shift	14056634	310164660	1291158810
Data Transfer	32692266	214161627	1636029315
Load/Store	11911031090	37507748969	81757176763
Others	534241804	1514087575	2581871188
Vectorization	5426852903	10429222578	268662027

(row=2,5,8)

(a): Before function specialization:

```
select case(mod(row,3))
 case 1:
  block(row+2,4:6,i1)=block(row+2,4:6,i1)-10
 case 2:
  block(row+1,1:3,i1)=block(row+1,1:3,i1)-10
end select
```

(b): After function specialization:

```
block(row+1,1:3,i1)=block(row+1,1:3,i1)-10
```

函数特化
→ 常数传播
→ 死码删除

编译优化机会识别（3）

Category	*digit_2.4*	*digit_2.7*	*MOD_digit_2*
Branch	3593142992	11609217450	29321533296
Indirect	50203	18775199	34934090
Jump	0	0	0
Call	0	0	0
Ret	50203	18775199	34934090
Cond	3361685141	10733730071	27698651020
Uncond Direct	231407648	856712180	1587948186
Operation	15199221930	41803072569	116664503804
Binary Arith	14686454611	39759170962	102813533806
Logical	498710685	1733736947	12559811188
Shift	14056634	310164660	1291158810
Data Transfer	32692266	214161627	1636029315
Load/Store	11911031090	37507748969	81757176763
Others	534241804	1514087575	2581871188
Vectorization	5426852903	10429222578	268662027

Source code:

```
block(row+1,4:6,i4)=block(
row+1,4:6,i4)+10
```

(a): 648.exchange2_s(BiSheng)

```
add   x8,x9,#0x32c
ldr   q0,[x8]
add   v0.4s,v0.4s,v3.4s
str   q0,[x8]
```

(b): 648.exchange2_s(GCC)

```
ldr   w20,[x14]
ldr   w19,[x14,#36]
ldr   w28,[x14,#72]
add   w27,w20,#0xa
add   w1,w19,#0xa
str   w27,[x14]
add   w8,w28,#0xa
str   w1,[x14,#36]
str   w8,[x14,#72]
```

向量化优化

其它“嫌疑热点”

Rank	BiSheng			GCC		
	relative time(%)	absolute time(sec)	function name	relative time(%)	absolute time(sec)	function name
1	**57.32**	**14.33**	***digits_2.7***	**82.38**	**32.05**	***MOD_digits_2***
2	**19.38**	**4.85**	***digits_2.4***	**7.30**	2.84	*gfortran_mminloc*
3	**18.30**	4.58	*logic_new_solver*	**4.15**	1.61	*specific.4*
4	1.46	0.37	*free*	**2.31**	0.90	*logic_MOD_new_solver*
5	0.80	0.20	*malloc*	1.08	0.42	*hidden_triplets.0*
6	0.43	0.11	*covered*	0.78	0.30	*free*
7	0.32	0.08	*brute*	0.55	0.21	*naked_triplets.1*
8	0.17	0.04	*f90_dealloc*	0.48	0.19	*hidden_pairs.2*
9	0.15	0.04	*f90_alloc*	0.25	0.10	*MOD_brute*
10	0.14	0.04	*f90_set_intrin*	0.25	0.10	*malloc*

编译优化机会识别（4）

Function	Complier	Vectorization	Ret
logic_new_solver	BiSheng	1963447369	22764
logic_MOD_new_solver	GCC	440253151	22088
specific.4	GCC	127785213	32759
gfortran_mminloc	GCC	0	24517022

函数内联 + 向量化优化

对GCC（AArch64）的优化建议

Benchmark	Optimization suggestions for GCC
605.mcf_s	(1) Optimize the capacity of inlining callback functions to eliminate indirect subroutine calls. (2) Optimize memory layout of the data structure, including structure peeling and unused field elimination.
648.exchange2_s	(1) Improve the optimization of inlining the non-tail recursive function calls. (2) Implement function specialization. (3) Optimize the capacity of automatic vectorization. (4) Inline the built-in functions for Fortran code.
625.x264_s	(1) Optimize the capacity of automatic vectorization. (2) Optimize the decisions of whether to inline a function.

优化效果验证

The performance improvement of our industry partner's GCC-based product compiler after implementing the identified optimizations.

Benchmark	before		after		improvement
	runtime(sec)	score	runtime(sec)	score	(score)
548.exchange2_r	527	636	306	1100	72.96%
505.mcf_r	1350	153	776	234	52.94%
525.x264_r	344	651	271	828	27.19%

Configurations:

- *Huawei Kunpeng 920-7261K(2 sockets, 128 cores in total;*
- *Operating System: CentOS Linux release 7.9.2009*
- *SPEC2017 input data: reference*

本次课程总结

- 通过程序插桩来获取应用负载运行过程中的动态行为是对负载进行特征分析以及性能优化的一个重要手段；
- 静态程序插桩和动态程序插桩有各自的优缺点；
- 二进制翻译技术可以大大助力程序插桩工具的实现；
- 精准插桩信息的高效收集、性能瓶颈的分析和准确定位、优化机会的快速识别以及优化机会的高效实现是软件系统优化永恒的研究课题。

补充材料

- An infrastructure for adaptive dynamic optimization (https://ieeexplore.ieee.org/document/1191551)
- Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation (https://dl.acm.org/doi/10.1145/1064978.1065034)
- BOLT: A Practical Binary Optimizer for Data Centers and Beyond (https://ieeexplore.ieee.org/document/8661201)
- OCOLOS: Online COde Layout OptimizationS (https://ieeexplore.ieee.org/document/9923868)
- Propeller: A Profile Guided, Relinking Optimizer for Warehouse-Scale Applications (https://dl.acm.org/doi/pdf/10.1145/3575693.3575727)


Q
Q
&
&
A
A