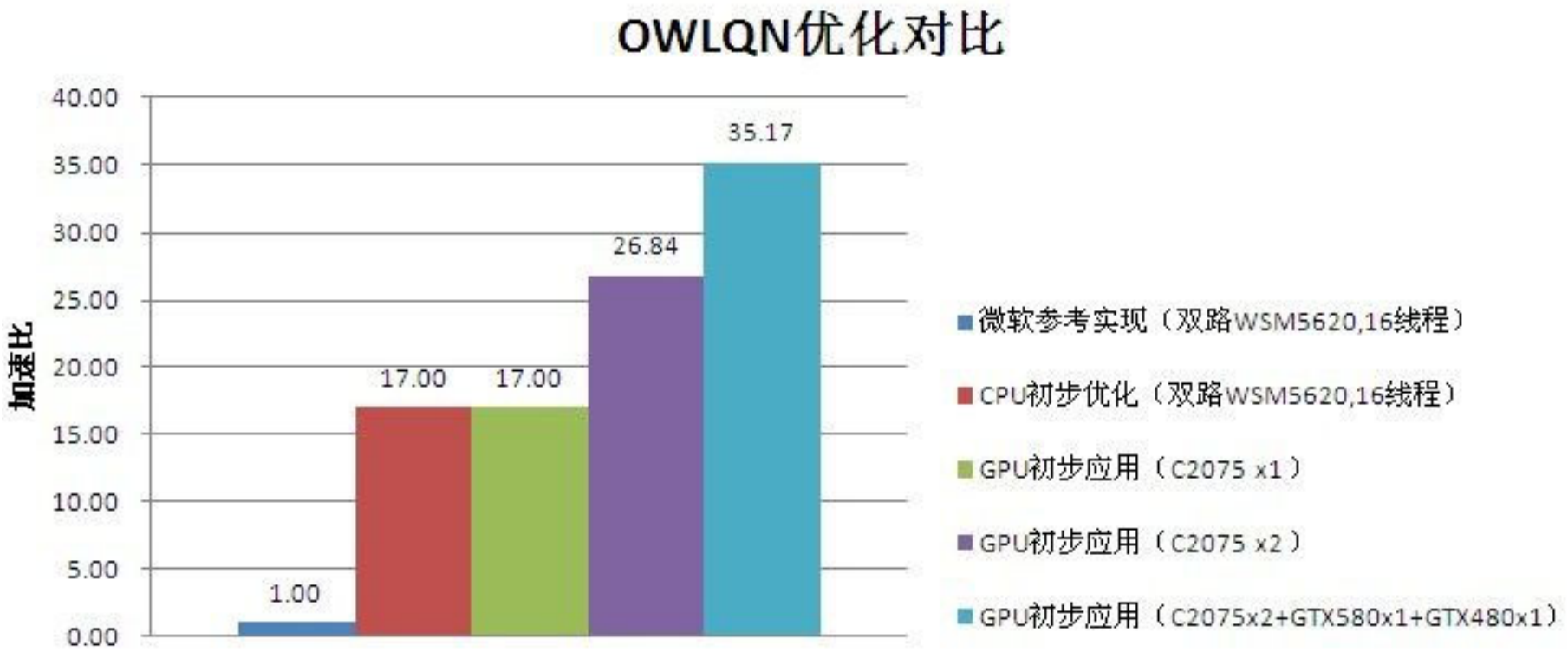
计算性能的极限

探求计算密集应用优化的天花板

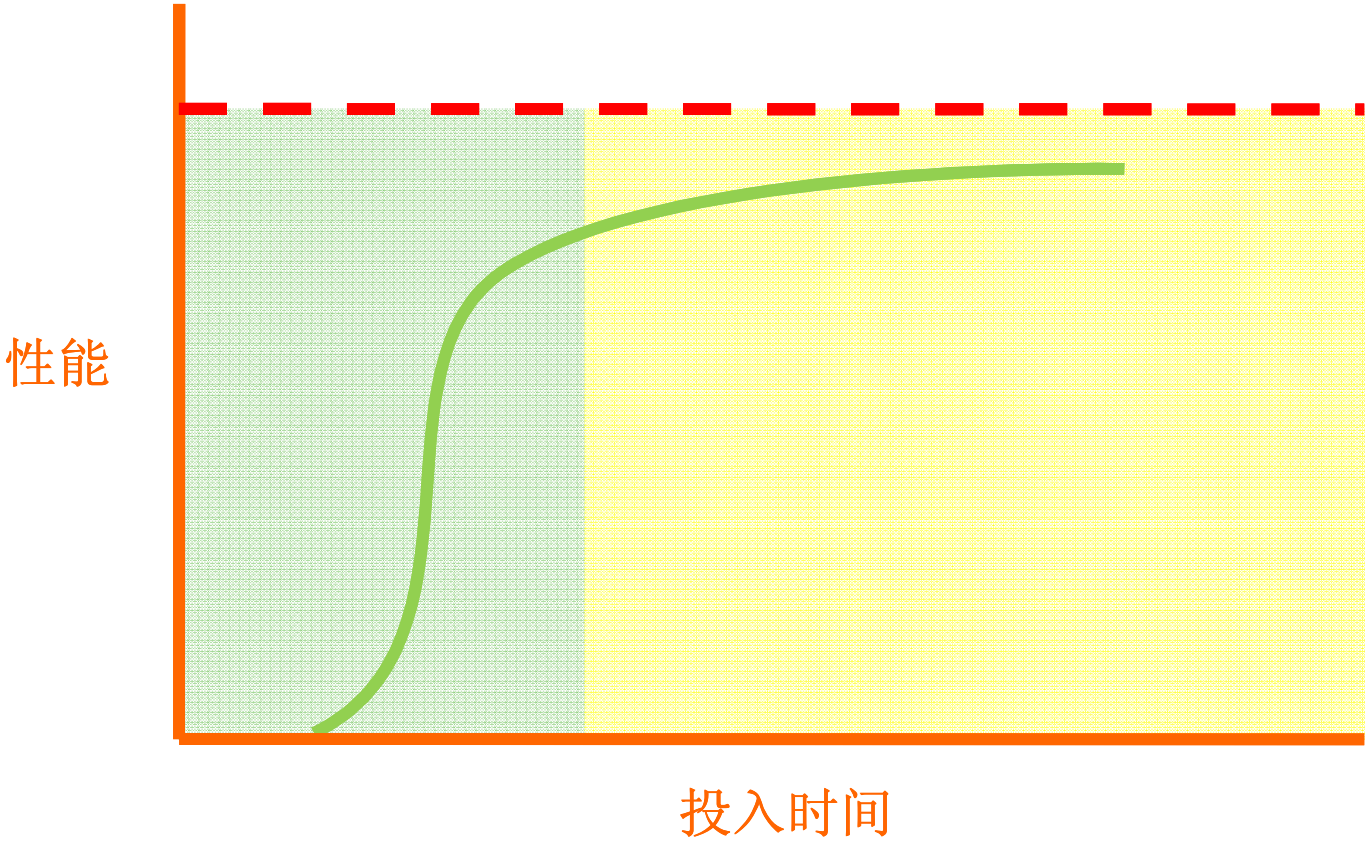
核心系统研发部 王琤

专用计算组的工作

- 针对特定硬件的“实现”优化
- 例：OWLQN算法CPU和GPU实现优化，[微软参考实现]



固定算法及平台下“投入加速比”曲线



红线在哪里

- 决定红线的因素复杂
  - 算法复杂度
  - 指令复杂度
  - 实际计算量
  - 访存复杂度
  - 等等
- 各种因素相互影响，决定了红线在哪

看不清的红线

- 不管你看的清看不清，反正它在那里

看的见的天花板

1. 天花板是各个维度上的平台极限性能
2. 确定的算法及实现决定了各个维度的“工作量”。
3. 特定硬件平台各个维度的天花板可以量化。
4. 由于2和3，决定了特定硬件平台，确定算法和实现的各个维度天花板可见。
5. 红线一定比所有天花板都低
6. 由于5推论：天花板最低者一定比红线高

看不清的红线

- “天花板最低者一定比红线高”等于：
  - 红线一定比天花板最低者还低
  - 接近天花板最低者也就接近红线了
  - 有幸撞到了天花板，恭喜你，这也是红线

撞到天花板是一件十分幸福的事

撞到天花板是一件十分幸福的事

- “超越极致前给先达到极致”
- 更改算法、实现，提升天花板才可能突破
- 尝试天花板更高的硬件
- 离天花板很远就止步了是最可怕的。
  - 离红线很近还是很远？
  - 没到“甜点”更悲催

我们常用体系结构某几个维度的天花板

- 通用CPU-Intel SNB
  - 指令执行速度
  - 计算能力
- GPGPU-Nvidia Fermi
  - 计算能力
- 众核-Intel Phi
  - 计算能力

声明

- 诺依曼体系指令、数据、执行。由于篇幅有限，通过忽略数据流体现理想计算性能极限。
- 所有示例都是以说明问题为目的最简单示例。
- 示例所示结果都经过实际验证。
- 示例虽然简单，但是都是针对各架构及微架构设计。平台变更，或者添加、删除或更改指令顺序都可能会导致关系性能指标的降低。
- 示例无实际意义，无厘头。

SNB指令的执行速度天花板

- 指令复杂度体现了算法复杂度
- 通用CPU指令级并行是有极限的
- 主流x86是4发射超标量
- 极致的指令级并行结果是每个时钟最多4 uops

无厘头示例1：CPU指令执行的极限

```
20:   xor    %r8,%r8
23:   xor    %r9,%r9
26:   xor    %r10,%r10
29:   sub    $0x1,%rax
2d:   jne    20
```

- Intel(R) Xeon(R) CPU E5-2680 (SNB)
- One iteration/cycle
- 4 uops/cycle
- IPC：5 ins/cycle CPI：0.2
- IPC/CPI只是参考

SNB纯计算性能的天花板

1. 执行单元和簇是有限的
2. 发射端口是有限的
3. 执行单元和端口分配各个微架构是固定的
4. 1、2、3决定了对于特定计算，不同微架构有理论峰值。例如SIMD浮点计算：

	Instruction Set	SP FLOPs per cycle per core	DP FLOPs per cycle per core
Nehalem	SSE (128-bits)	8	4
Sandy Bridge	AVX (256-bits)	16	8

无厘头示例2：CPU浮点极限

```
60:     vaddps      %ymm2,%ymm2,%ymm1
64:     vmulps      %ymm4,%ymm4,%ymm3
68:     sub         $0x1,%rax
6c:     jne         60
```

- Intel(R) Xeon(R) CPU E5-2680 (SNB)
- One iteration/cycle
- 3个端口全部利用
- 8 SP add+8 SP mul/cycle/core
- 16 SP ops*8cores*2 ways*2.7GHZ/s=691G SP Flops
- 注意值，避免出现underflow等问题

无厘头示例3：CPU定点极限

```
70:     psubb       %xmm9,%xmm9
75:     pmuludq     %xmm9,%xmm9
7a:     paddq       %xmm8,%xmm8
7f:     sub         $0x1,%rax
83:     jne         70
```

- Intel(R) Xeon(R) CPU E5-2680 (SNB)
- 不用AVX，2 operands要到极致需要稍微多考虑一点点
- One iteration/cycle
- 3个计算端口全部利用
- (128bits(packed 64 bits) mul+128bits(packed 64 bits) mul+64bits sub)/cycle/core
- 320 bits定点/cycle

无厘头示例4：CPU定点极限2

```
c0:     psubb       %xmm9,%xmm9
c5:     pmuludq     %xmm9,%xmm9
ca:     paddq       %xmm10,%xmm10
cf:     paddq       %xmm8,%xmm8
d4:     psubb       %xmm7,%xmm7
d8:     pmuludq     %xmm7,%xmm7
dc:     paddq       %xmm6,%xmm6
e0:     sub         $0x1,%rax
e4:     jne         c0
```

- Intel(R) Xeon(R) CPU E5-2680 (SNB)
- One iteration/2 cycle
- 3个计算端口全部利用
- (2x128bits(packed 64 bits) mul+3x128bits(packed 64 bits) add+64bits sub)/2 cycle/core
- 352 bits定点/cycle。不断减少循环所需port5，利用率不断趋向384 bits/cycle

Phi计算性能

- P54c的改造，利用成熟的“简单核”
- 双发射，U，V流水线
- 支持64位扩展
- 512 bits SIMD扩展，在V上。
- 用它做标量并行？用错了，标量计算理论上讲，61颗Pentium@1.1G，约为两颗E5-2680@2.7G。
- 感谢工艺提升，当年不到三百万晶体管的P54核心搞几十个放到50亿晶体管的Phi核中，只占很小的比例。

无厘头示例5： Phi双精度峰值

```
60：vfmadd231pd %zmm18, %zmm19, %k2, %zmm21
66：sub $0x1, %rax
6a：jnz 60
```

- Xeon Phi coprocessor SE10，61核@1.1G
- 利用fma实现峰值
- One iteration/2 cycle
- 单线程Vfmadd231pd吞吐1 cycle+1 cycle的前端stall
- 所以单线程极限： 16 df/2 cycle。
- 必须利用多线程达到峰值，每核4线程。2线程以上实现16df/cycle/core
- 16dfx60x1.1G=1.05T double flops

Fermi计算性能

- 简单核心，超多计算执行单元，c2075 448个cuda core。
- 每个核心单发射超标量。
- 计算流水线几级，整体几十级。
- 更多的线程掩盖整体流水线、计算执行、访存等诸多延迟。
- 单精度理论，不算SF，1030.4GFLOPS
- 双精度理论，不算SF, 515.2GFLOPS
- 单双精度需要FMA才能实现。

无厘头示例6：Fermi单精度峰值

```
S2R             R0, SR_Tid_X;
I2F.F32.U32     R2, R0;
LOP.AND         R0, R0, 0xf;
FFMA.FTZ.RZ     R2, R2, R2, R2;
SHL.W           R0, R0, 0x2;
FFMA.FTZ.RZ     R2, R2, R2, R2;
[Unroll]
FFMA.FTZ.RZ     R2, R2, R2, R2;
STS             [R0], R2;
```

- Nvidia Tesla C2075
- 利用FMA达到峰值
- 1024 threads/block,140 blocks实测928.7G flops
- 非FMA单精度计算极限减半

无厘头示例7：Fermi双精度峰值

```
S2R             R0, SR_Tid_X;
I2F.F64.U32     R2, R0;
LOP.AND         R0, R0, 0xf;
DFMA.RZ         R2, R2, 0x40000, R2;
SHL.W           R0, R0, 0x3;
DFMA.RZ         R2, R2, 0x40000, R2;
[Unroll]
DFMA.RZ         R2, R2, 0x40000, R2;
STS.64          [R0], R2;
```

- Nvidia Tesla C2075
- 利用FMA达到峰值
- 1024 threads/block,140 blocks实测489G flops
- 非FMA双精度计算极限减半

对比

- 通用CPU
  - 关注向增加指令级并行方向发展
  - 结合数据级并行和线程级并行
  - 单个核非常复杂
  - 理论计算性能不低，下一代Haswell将支持FMA，翻翻。
- GPGPU和众核
  - 每个核很简单，数量多
  - 关注通过线程级并行或数据级并行
  - 突出吞吐量
  - 理论计算性能很高。

我们的经验：现实中的红线和天花板

- 算法和实现特征决定
  - 计算密集
  - 访存密集
  - 指令密集
- 目前接触到的普遍应用访存密集，计算访存比不是非常高。
- “怕跑不怕算”，footprint和局部性往往是更多情况下关注的。
- 算法实现整体红线离天花板可能比较远
- 局部关键kernel容易碰到天花板
- 局部往上顶一般都没有坏处

我们的经验：算法和实现的关系

- 算法和实现是双轨
- 算法复杂度提高但是实现更快的例子很多
- 实现影响算法
  - 数据结构
  - 处理流程
  - 细节算法
  - 等等

我们的经验：编译器的角色

- 领导和秘书的关系
- 码指令的方法众多，成本收益比不同
  - 自动并行化
  - 自动向量化
  - ……
  - 汇编
- 编译器能够完美做好优化的那天理论上它也能自动编程了
  - 太笨，bug
  - 自作聪明，沟通问题

我们的经验：想象力是王道

- 此处留白，亲，想象去吧

总结

- 计算优化的红线很重要。
- 天花板和红线之间有关系，天花板是可见的。
- 用无厘头示例展示了目前我们用的通用CPU，GPGPU，众核计算性能的天花板。
- 分享了我们做实现优化方面的一些经验。

没有广告的分享不是好广告

- 专用计算组欢迎内部转岗及社招推荐
  - 计算优化
    - 针对特定计算密集型应用，及特定体系架构算法及实现优化。
    - 目前进行中项目包括图像搜索算法、机器学习实现算法等。
  - 阿里JVM定制优化，针对阿里Java应用特点的
    - 性能优化
    - 功能增强
    - Bug修复
- 邮箱：changren@taobao.com 微博：@王王争

谢谢

- 问答时间