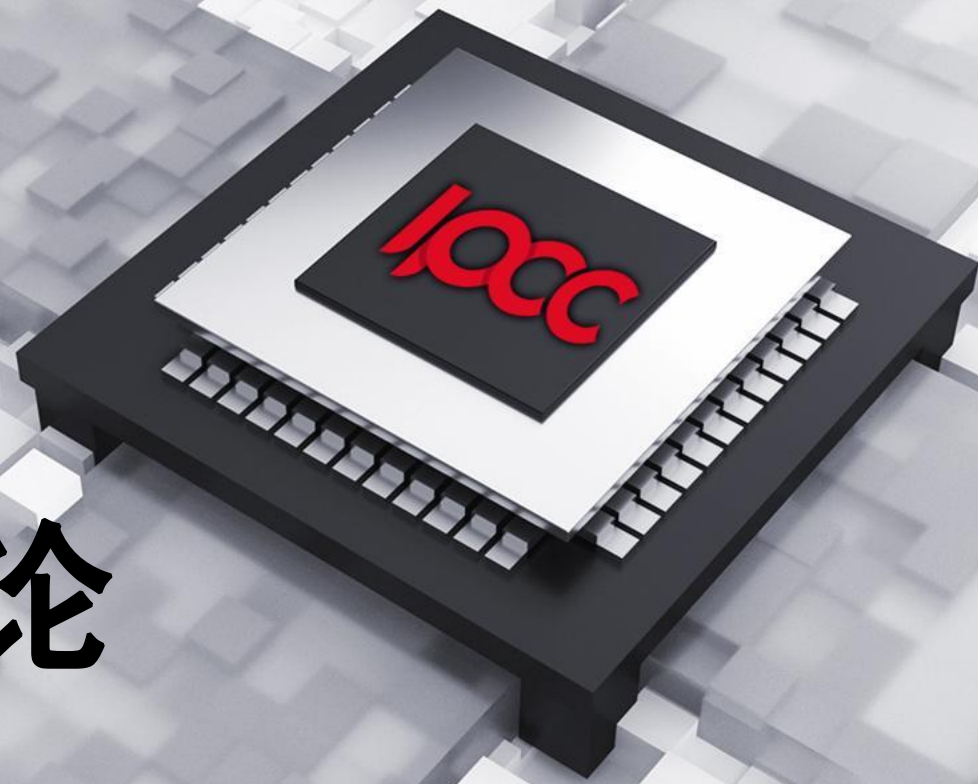


现代处理器优化概论

ACM 中国-国际并行挑战赛-赛前讲座2022

第 3 讲 / 共 6 讲

i@wu-kan.cn
https://wu-kan.cn

中山大学超算队 吴坎

中山大學 SUN YAT-SEN UNIVERSITY
国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

本届讲座大纲

Association for Computing Machinery

ACM中国 国际并行计算挑战赛

主讲人：吴坎

- 第一讲：超算竞赛备赛指南
  - 直播时间：05月26日 周四
- 第二讲：超算基本原理与方法论
  - 直播时间：06月~~01~~02日 周四
- **第三讲：现代处理器优化概论**
  - 直播时间：06月08日 周三

主讲人：叶子凌锋

- 第四讲：并行开发技术与优化方法 I
  - 直播时间：06月17日 周五
- 第五讲：并行开发技术与优化方法 II
  - 直播时间：06月24日 周五
- 第六讲：并行优化常用方法与工具
  - 直播时间：06月30日 周五

往期分享回顾

更多学习资料与往届
讲座分享，尽在比赛
交流群
1046805935 ！

学习平台：哔哩哔哩[超级云讲堂]

- 第一讲：并行计算基础概论 | 主讲人：邵奇
- 第二讲：性能优化方法论 | 主讲人：赵雄君
- 第三讲：并行开发技术概论 | 主讲人：张力越
- 第四讲：现代处理器优化技术 | 主讲人：邵奇
- 第五讲：并行优化常用工具 | 主讲人：邵奇
- 第六讲：并行优化实战技巧 | 主讲人：张力越

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

超算基本原理与方法论

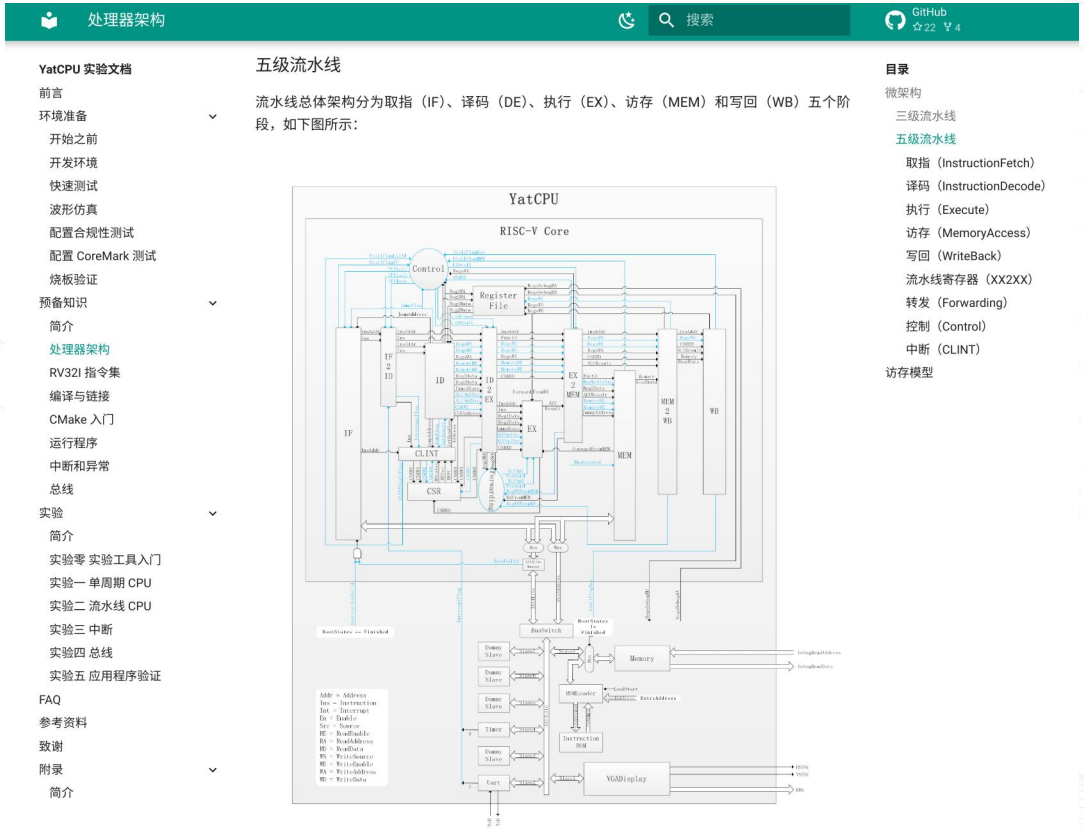
- 超级计算机基本原理
  - 摩尔定律与十年千倍定律
  - Amdahl 定律与 Gustafson 定律
- 超算技术发展史与方法论（软件、硬件、理论协同发展）
  - 60～70 年代：FORTRAN/C/C++ 语言、集成电路、Flynn 分类法
  - 70～80 年代：线性代数过程（BLAS、Linpack）、向量处理（SIMD）
  - 90 年代：共享内存（SMP、OpenMP）、消息传递（MPP、MPI）
  - 00 年代：大规模集群计算（Cluster）
  - 10 年代：异构计算（CUDA、ROCm、OpenCL 等）
- 超算经典使用案例与展望

现代处理器优化概论

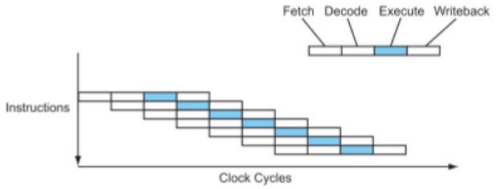
- “现代”处理器重要概念
  - 我们为什么要了解处理器
  - 我们要从哪些维度了解处理器
- “现代”处理器简单模型
  - 取指、译码、重命名、发射、执行、写回、提交…
- 优化妙妙屋
  - 指令级并行（ILP）
    - 数据依赖、循环依赖、循环展开…
  - 访存优化
    - 分支预测、数据对齐、局部性优化…

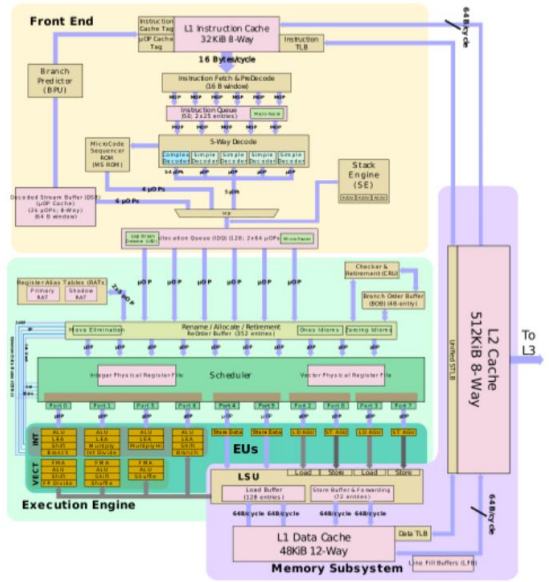
我们为什么要了解处理器

- 处理器是如何认知我们的程序的?
  - 建立自底而上的全局观!
- “现代处理器”与经典教学用流水线处理器的异同?



https://yatcpu.sysu.tech/tutorial/arch/





左边这个大概上过计组的都懂，给它加亿点点细节就成了右边这样啦!

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

要从哪些维度了解处理器?

对于不同架构的处理器，这些答案都是“No”！

- 处理器的频率？频率更高的处理器性能一定更好吗?
- 是否使用流水线设计？流水级数越长越好吗?
- 顺序执行/乱序执行?
  - 乱序时可不按指令顺序执行以减少阻塞，但结果仍与顺序执行相同
- 标量处理器/超标量处理器?
  - 标量处理器只有一套执行单元，IPC 最多为 1，否则为超标量处理器
- 向量处理器?
- 多线程处理器？多核处理器?
  - 超线程，多个线程共享相同物理资源

要从哪些维度了解处理器?


```
[sc90194@ln112%bscc-a3 ipcc2021-final]$ cat cpu.sh
#!/bin/bash
#SBATCH -p amd_256
#SBATCH -N 1
#SBATCH --exclusive
cat /proc/cpuinfo
cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
[sc90194@ln112%bscc-a3 ipcc2021-final]$ sbatch cpu.sh
Submitted batch job 2526491
[sc90194@ln112%bscc-a3 ipcc2021-final]$ cat slurm-*.out | tail -n 29

processor       : 63
vendor_id       : AuthenticAMD
cpu family      : 23
model           : 49
model name      : AMD EPYC 7452 32-Core Processor
stepping        : 0
microcode       : 0x8301052
cpu MHz         : 2345.747
cache size      : 512 KB
physical id     : 1
siblings        : 32
core id         : 31
cpu cores       : 32
apicid          : 95
initial apicid  : 95
fpu             : yes
fpu_exception   : yes
cpuid level     : 16
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm con
stant_tsc art rep_good nopl nonstop_tsc extd_apicid aperfmperf eagerfpu pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdran
d lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_l2 cpb cat_l3 cdp
_l3 hw_pstate sme retpoline_amd ssbd ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 cqm_
llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr wbnoinvd arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfth
reshold avic v_vmsave_vmload vgif umip overflow_recov succor smca
bogomips        : 4691.65
TLB size        : 3072 4K pages
clflush size    : 64
cache_alignment : 64
address sizes   : 43 bits physical, 48 bits virtual
power management: ts ttp tm hwpstate cpb eff_freq_ro [13] [14]

     64  AMD EPYC 7452 32-Core Processor
[sc90194@ln112%bscc-a3 ipcc2021-final]$
```

i@wu-kan.cn
https://wu-kan.cn

中山大学超算队 吴坎

要从哪些维度了解处理器?

- flops=核数*单核主频*单周期浮点计算次数
  - 单节点两路 EPYC 7452，每路 32 物理核，共 64 核
  - 单核频率 2.35 GHz，最高可睿频至 3.35 GHz，是否 bios 中锁频?
  - 单周期浮点计算次数怎么求?
    - 单核一个/两个 avx2 单元（查文档、架构图）
    - avx2 单元处理宽度 256 bit，即单次指令可做 256/32 = 8 个单精度 fma 运算
    - 1 个 fma 运算包含一个 mul、一个 add，等效 2 次浮点运算
- 每路带宽约 200GB/s（3200MHz/2*64/8*2*8）
  - 实测每路约 140 GB/s
- 实测为何很难达到理论算力/带宽?

AMD EPYC™ 7452

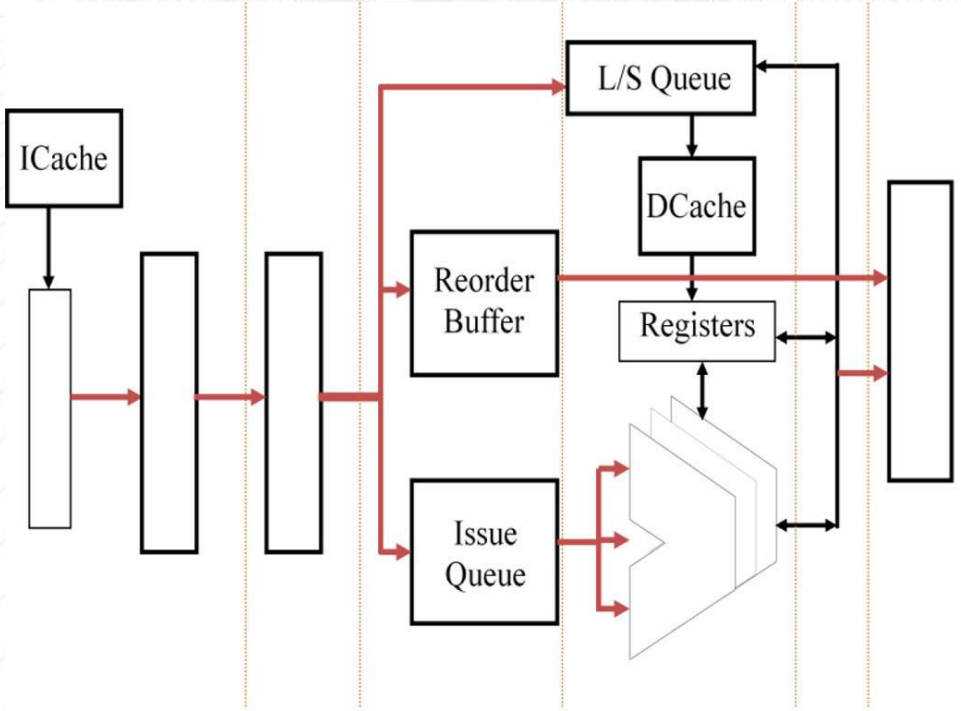
General Specifications	平台: 服务器	产品家族: AMD EPYC™ (霄龙)
	产品系列: AMD EPYC™ 7002 Series	CPU 核心数量: 32
	线程数量: 64	最大加速时钟频率: 最高可达 3.35GHz
	基准时钟频率: 2.35GHz	三级缓存: 128MB
	默认 TDP/TDP: 155W	封装: SP3
	支持的CPU插槽数: 1P/2P	
Connectivity	PCI Express 版本: PCIe 4.0 x128	内存类型: DDR4
	内存通道: 8	最高内存速度: 最高可达3200MHz
	内存带宽（每路）: 204.8 GB/s	

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

“现代处理器”运行流程

一些对性能要求不高，而对能耗和芯片面积敏感的场景下，处理器的设计则可能会简化为三级流水线甚至更精简，此处不考虑

- 取指
  - 从内存获取下条待执行指令
- 译码
  - 将指令解码为相应控制指令
- 重命名
  - 给需要执行的指令分配寄存器
  - 将指令分配到一条发射队列
- 发射
  - 检查队列中指令的操作数是否就绪
  - 将就绪的指令发送到执行单元执行

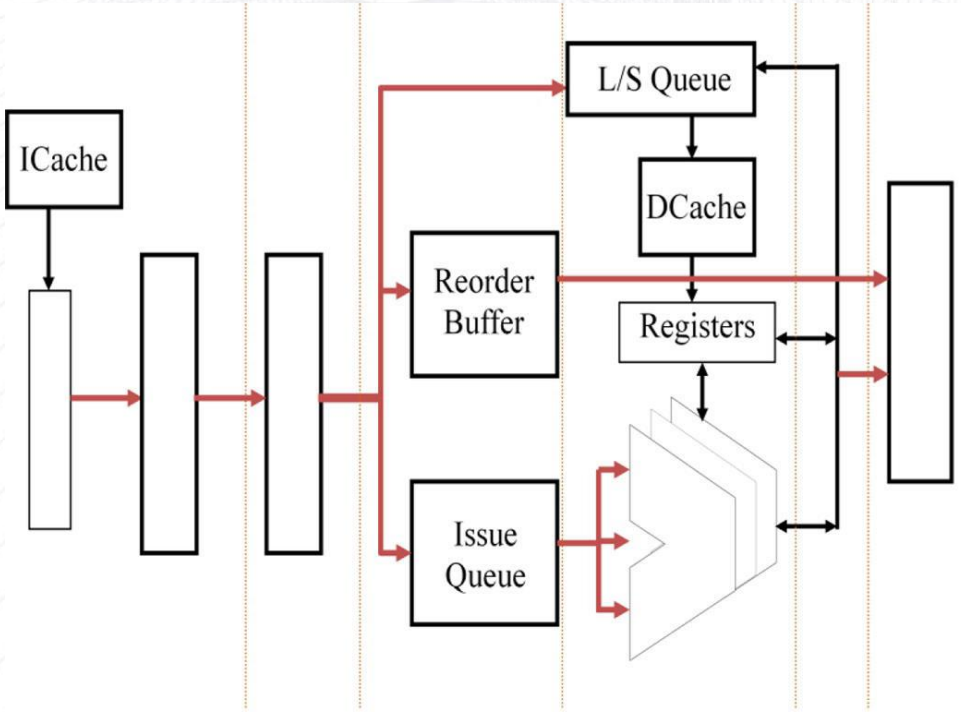


《Processor Microarchitecture：An Implementation Perspective》

“现代处理器”运行流程

一些对性能要求不高，而对能耗和芯片面积敏感的场景下，处理器的设计则可能会简化为三级流水线甚至更精简，此处不考虑

- 执行
  - … 理论算力只考虑了“执行”阶段！
  - 优化：尽可能增加“执行”比重
- 写回
  - 将执行结果写回寄存器
  - （乱序处理器）写回重排缓冲区
- 提交
  - 完成一条指令
  - 修改处理器内部状态



《Processor Microarchitecture：An Implementation Perspective》

增加ILP（指令级并行）

- 准确定义指令级并行
  - 指令之间不存在相关
  - 在流水线中存在重叠执行的潜在并行性
- 静态 ILP
  - 手动或利用编译器对指令进行静态调度
  - 以减少数据依赖与冲突
- 动态 ILP
  - 处理器硬件发现并行机会
- How?

天气冷了，进来坐坐吧

数据依赖、循环依赖

- 数据依赖、循坏依赖
  - 减少乱序执行机会，降低 ILP
  - 也使循环难以直接被并行

> • Consider the following loop, Are there dependences between S1, S2 and S3? Is this loop parallel? If not, show how to make it parallel.

```
for (int i = 0; i < 100; ++i) {
  A[i] = B[i] + C[i];     /* S1 */
  B[i + 1] = D[i] + E[i]; /* S2 */
  C[i + 1] = D[i] * E[i]; /* S3 */
}
```

存在循环依赖，S1 依赖前一轮的 S2、S3。可以通过调整语句顺序，去掉循环依赖，效果如下。

```
A[0] = B[0] + C[0];
#pragma omp parallel for
for (int i = 0; i < 99; ++i) {
  B[i + 1] = D[i] + E[i];
  C[i + 1] = D[i] * E[i];
  A[i + 1] = B[i + 1] + C[i + 1];
}
B[100] = D[99] + E[99];
C[100] = D[99] * E[99];
```

> • In the following loop, find all the true dependencies, output dependencies, and anti-dependencies. Eliminate the output dependencies and anti-dependencies by renaming.

```
for (int i = 0; i < 100; ++i) {
  A[i] = A[i] * B[i]; /* S1 */
  B[i] = A[i] + c;    /* S2 */
  A[i] = C[i] * c;    /* S3 */
  C[i] = D[i] * A[i]; /* S4 */
}
```

- 输出相关（写后写）：S1 和 S3 通过 A[i] 产生了输出相关。
- 反相关（读后写）：S3 和 S1 通过 A[i]；S2 和 S1 通过 B[i]；S3 和 S2 通过 A[i]；S4 和 S3 通过 C[i] 产生了反相关。
- 真数据相关（写后读）：S2 和 S1 通过 A[i]；S4 和 S3 通过 A[i] 产生了真数据相关。

改写代码，通过重命名去掉输出相关和反相关：

```
for (int i = 0; i < 100; ++i) {
  A[i] = A[i] * B[i];   /* S1 */
  B1[i] = A[i] + c;     /* S2 */
  A1[i] = C[i] * c;     /* S3 */
  C1[i] = D[i] * A1[i]; /* S4 */
}
```

https://wu-kan.cn/2021/12/03/HW3/

循环展开

循环展开

- 进行多路循环展开，可以极大地减少分支预测次数
- 对于较短的计算内核，进行循环展开可以极好的提升指令流水

```
#define M 1000
int loop() {
  //...
  for (int i = 0; i < M; i++) {
    c[i] = a[i] + b[i];
  }
  //...
}
```

```
#define N 4
int unroll_loop() {
  //...
  int i;
  for (i = 0; i < M - (N - 1); i+=N) {
    c[i] = a[i] + b[i];
    c[i + 1] = a[i + 1] + b[i + 1];
    c[i + 2] = a[i + 2] + b[i + 2];
    c[i + 3] = a[i + 3] + b[i + 3];
  }
  for (; i < M; i++) {
    c[i] = a[i] + b[i];
  }
  //...
}
```

- 手动展开 or 自动展开
  - #pragma unroll
- 优势
  - 减少分支跳转（作用没那么大，处理器分支预测准确率超过 95%）
  - 增加乱序执行的机会，提升指令并行度
  - 增加向量化（手动/编译器自动）的机会，提升数据并行度
- 展开到什么程度比较合适？越多越好吗？
  - [LCTES'2022] T. Ge, Z. Mo, K. Wu, X. Zhang and Y. Lu. RollBin: Reducing Code-size via Loop Rerolling at Binary Level

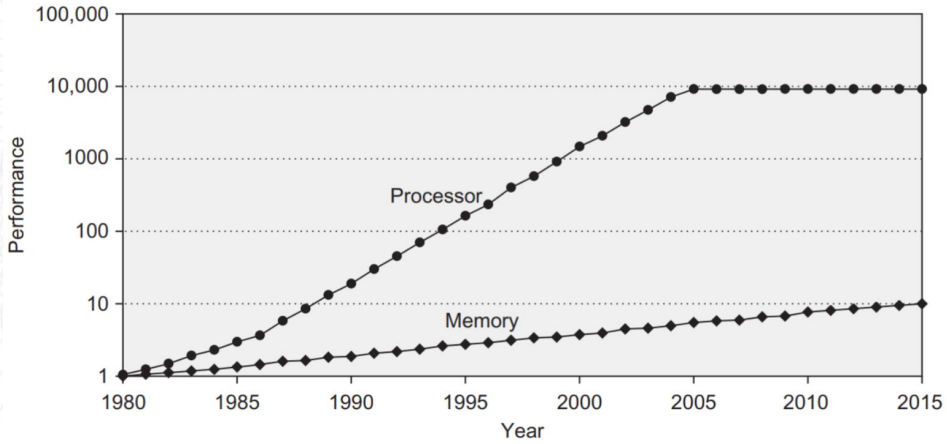
访存优化

- 存储墙
  - 处理器性能发展速度远快于存储器
- 访存密集应用仍然很多
- 层次化存储器结构
  - 兼顾空间、性能、性价比
  - 如何利用？

10 Advanced Optimizations

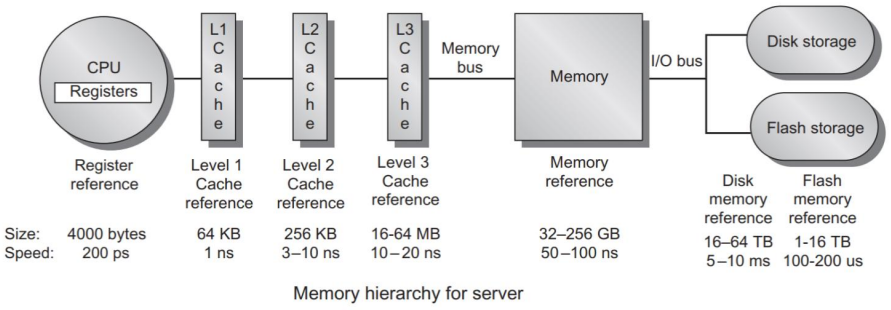
- **Reduce hit time**
  - 1. Small and simple first-level caches
  - 2. Way prediction
- **Increase bandwidth**
  - 3. Pipelined caches,
  - 4. multi-banked caches,
  - 5. non-blocking caches
- **Reduce miss penalty**
  - 6. Critical word first
  - 7. merging write buffers
- **Reduce miss rate**
  - 8. Compiler optimizations
- **Reduce miss penalty or miss rate via parallelization**
  - 9. Hardware prefetching
  - 10. compiler prefetching

Processor-Memory Performance Gap



Memory Hierarchy

- **Memory Hierarchy for a Server**



(C) Memory hierarchy for server

分支预测

- 写让处理器更容易预测的代码
  - 减少 Cache Miss
- 显式：编译指示
- 隐式：使数据有序
  - 处理数据前先排个序，可能有奇效

Way Prediction

- **To improve hit time, predict the way of the next cache access**
  - Multiplexor is set early to select the desired block
  - In that clock cycle, a single tag comparison is performed in parallel with reading the cache data.
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- **Extend to predict block as well**
  - Also called "Way selection"
  - Increases mis-prediction penalty

分支预测

- 程序员可以显示告诉编译器某一分支的概率，以辅助编译器进行代码优化

```
#define likely(exp)     __builtin_expect(!!(exp), 1)
#define unlikely(exp)   __builtin_expect(!!(exp), 0)

#define likely_prob(exp, probability) \
  __builtin_expect_probability(!!(exp), 1, probability)
#define unlikely_prob(exp, probability) \
  __builtin_expect_probability(!!(exp), 0, probability)

int main(int argc, char* argv[]) {
  int a;
  a = atoi(argv[1]);
  if (a > 1) {
    a += 1;
  } else {
    a -= 8;
  }
  printf("a: %d\n", a);
  return 0;
}
```

分支预测

预测前：

```
0000000000400490 <main>:
  400490: 48 83 ec 08     sub    $0x8,%rsp
  400494: 48 8b 7e 08     mov    0x8(%rsi),%rdi
  400498: ba 0a 00 00 00  mov    $0xa,%edx
  40049d: 31 f6           xor    %esi,%esi
  40049f: e8 dc ff ff ff  callq  400480 <strtol@plt>
  4004a4: 8d 50 f8        lea    -0x8(%rax),%edx
  4004a7: 8d 70 01        lea    0x1(%rax),%esi
  4004aa: 83 f8 02        cmp    $0x2,%eax
  4004ad: bf 40 06 40 00  mov    $0x400640,%edi
  4004b2: 0f 4c f2        cmovl  %edx,%esi
  4004b5: 31 c0           xor    %eax,%eax
  4004b7: e8 94 ff ff ff  callq  400450 <printf@plt>
  4004bc: 31 c0           xor    %eax,%eax
  4004be: 48 83 c4 08     add    $0x8,%rsp
  4004c2: c3              retq
```

预测后：

```
0000000000400490 <main>:
  400490: 48 83 ec 08     sub    $0x8,%rsp
  400494: 48 8b 7e 08     mov    0x8(%rsi),%rdi
  400498: ba 0a 00 00 00  mov    $0xa,%edx
  40049d: 31 f6           xor    %esi,%esi
  40049f: e8 dc ff ff ff  callq  400480 <strtol@plt>
  4004a4: 83 f8 01        cmp    $0x1,%eax
  4004a7: 7e 16           jle    4004bf <main+0x2f>
  4004a9: 8d 70 01        lea    0x1(%rax),%esi
  4004ac: bf 40 06 40 00  mov    $0x400640,%edi
  4004b1: 31 c0           xor    %eax,%eax
  4004b3: e8 98 ff ff ff  callq  400450 <printf@plt>
  4004b8: 31 c0           xor    %eax,%eax
  4004ba: 48 83 c4 08     add    $0x8,%rsp
  4004be: c3              retq
  4004bf: 8d 70 f8        lea    -0x8(%rax),%esi
  4004c2: eb e8           jmp    4004ac <main+0x1c>
```

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

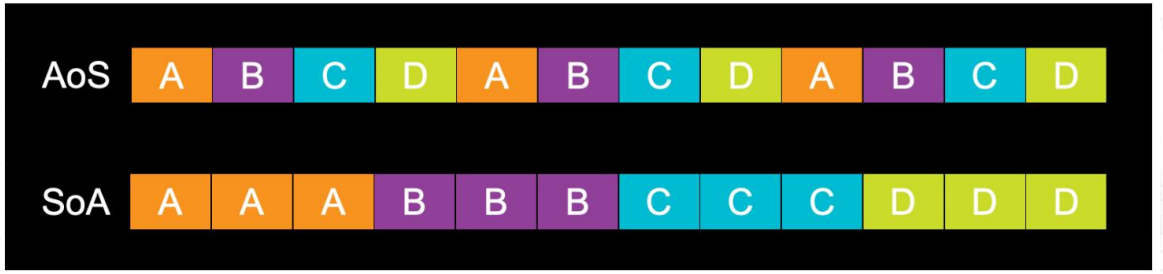
数据对齐

- 减少访存开销
- AOS VS SOA?
  - 后者局部性略差
  - 但访存更加规整，便于 SIMD

```
struct AoSData
{
    public int a;
    public int b;
    public int c;
    public int d;
}

struct SoAData
{
    public NativeArray<int> aArray;
    public NativeArray<int> bArray;
    public NativeArray<int> cArray;
    public NativeArray<int> dArray;
}
```

他们的数据在内存中的布局就如下图：



https://developer.unity.cn/projects/61ff5161edbc2a001cf9856e

数据对界

- 向量化指令在装载及存储数据时往往需要进行数据对界，已产生更好的存取性能
- 以AVX2为例，load及store指令需要32位对界 或 使用非对界指令

```
int *a_base = (int *)malloc(sizeof(int) * M + 64);
int *a = (int *)(((long)a_base + 0x40) & (~0x3f));
```

```
for (int i = 0; i < M; i+=8) {
  __m256i va, vb, vc;
  va = _mm256_loadu_si256(a + i);
  vb = _mm256_loadu_si256(b + i);
  vc = _mm256_add_epi32(va, vb);
  _mm256_storeu_si256(c + i, vc);
}
```

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

局部性优化

- Row Major vs Col Major
  - 按需使用
- 合理选择循环遍历顺序
  - 可考虑分块处理

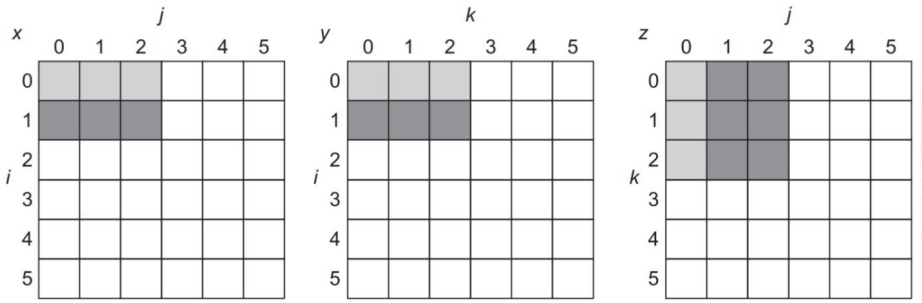
- **Blocking**
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
    - The goal is to maximize accesses to the data loaded into the cache before the data are replaced.
  - Requires more memory accesses but improves locality of accesses
  - improves temporal locality to reduce misses

- **Loop Interchange**
  - Swap nested loops to access memory in sequential order
    - The original code would skip through memory in strides of 100 words
    - The revised version accesses all the words in one cache block before going to the next block

```
/* Before */
for (j = 0; j < 100; j = j + 1)
      for (i = 0; i < 5000; i = i + 1)
            x[i][j] = 2 * x[i][j];
/* After */
for (i = 0; i < 5000; i = i + 1)
      for (j = 0; j < 100; j = j + 1)
            x[i][j] = 2 * x[i][j];
```

```
/* After */
for (jj = 0; jj < N; jj = jj + B)
for (kk = 0; kk < N; kk = kk + B)
for (i = 0; i < N; i = i + 1)
      for (j = jj; j < min(jj + B,N); j = j + 1)
          {r = 0;
           for (k = kk; k < min(kk + B,N); k = k + 1)
               r = r + y[i][k]*z[k][j];
           x[i][j] = x[i][j] + r;
          };
```

B = 3



i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

不讲武德的奇技淫巧

- 用高效位运算代替乘除法运算，提高“执行”过程的效率
  - 丧失可读性，有时可交给编译器
- 使用内建函数，更加高效/对编译器做出“提示”
  - 如，手动数据预取或分支概率提示
- 内联汇编，精细控制代码的每个细节
  - 破坏编译器前后依赖分析
- 多跑几次，热身运行…

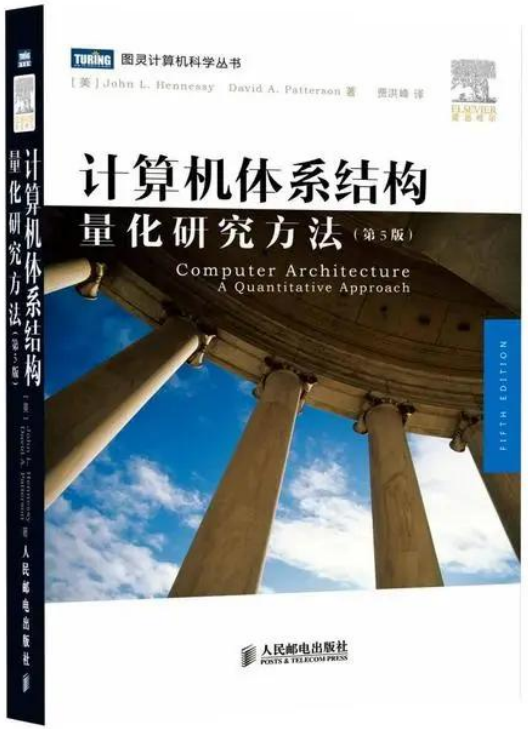
前三讲总结

每一讲都有一个贯穿的线索

- 第一讲：超算竞赛备赛指南
  - 自始至终：如何进行一次完整的代码优化？
- 第二讲：超算基本原理与方法论
  - 自顶向下：如何进行大规模并行应用开发？
- 第三讲：现代处理器优化概论
  - 自底而上：“现代”处理器怎样认知我们的程序？
- 多多实践，方能不囿于“技”，游刃有余！
  - *“道也，进乎技矣。”* ——*《庖丁解牛》*

i@wu-kan.cn
https://wu-kan.cn
中山大学超算队 吴坎

扫码提交问卷调查



组委会联系方式

- 官网：www.paraedu.org.cn
- 邮箱：ACM_IPCC@163.com
- 电话：18310726311（余老师）
- 微信公众号：
  - 北京超级云计算中心(BJBLSC)
- 交流群：
  - 1046805935（参赛选手）
  - 1095416620（指导老师）

Thanks!

ACM 中国-国际并行挑战赛-赛前讲座2022

下期预告：并行开发技术与优化方法 I

不要走开，精彩继续~

i@wu-kan.cn
https://wu-kan.cn

中山大学超算队 吴坎

中山大學 SUN YAT-SEN UNIVERSITY 国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU