计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(6):56−64 [doi: 10.15888/j.cnki.csa.008508]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于通用多核处理器的 5G 应用并发场景行为分析①

张亚琳, 李　浩, 胡　骁, 梁　卓, 潘彦鹏

(中国航天科技集团有限公司第一研究院 战术武器总体技术部, 北京 100076)
通信作者: 张亚琳, E-mail: 1055069370@qq.com

摘　要: 由于 5G 通信场景具有大带宽、低延迟、海量流量和多样性等特征, 5G 业务由传统基站承载转向数据中心部署已成为趋势. 为给出 5G 应用在数据中心的部署建议, 以开源 OpenLTE 作为其代表性 benchmark 进行分析, 由于开源的 OpenLTE 性能很差, 在性能分析时不能反映真实场景和行为特征, 因此首先根据通用处理器的特点对其代码进行了优化, 取得了 2.5 倍性能加速比; 在此基础上结合处理器特征分析应用执行行为, 其主要特点为: 5G 下行过程物理层处理流程是计算密集的, 最高端口利用率 90%, 访存不密集, 程序响应时间极短; 最后结合通用处理器独特参数 (多核、SMT 和 Turbo Boost 等) 分析应用在并发场景下的行为表现, 并以提升数据中心资源率为目的给出部署建议, 5G 应用的强实时性使其只能以独占机器方式运行, 其内部并发体之间对共享缓存和访存带宽竞争小而对执行部件竞争激烈, 可采用并发量不多于处理器核数方式部署, 同时 TurboBoost 的影响不可忽视.

关键词: 通用处理器; 5G; OpenLTE; 行为分析; 代码优化

引用格式: 张亚琳,李浩,胡骁,梁卓,潘彦鹏.基于通用多核处理器的 5G 应用并发场景行为分析.计算机系统应用,2022,31(6):56–64. http://www.c-s-a.org.cn/1003-3254/8508.html

Analysis of 5G Workload's Performance Based on General Multi-core Processor in Concurrent Scenario

ZHANG Ya-Lin, LI Hao, HU Xiao, LIANG Zhuo, PAN Yan-Peng

(Department of Tactical Weapon General Technology, China Academy of Launch Vehicle Technology, Beijing 100076, China)

Abstract: Due to the characteristics of large bandwidth, low latency, massive traffic, and diversity in 5G communication scenarios, it has become a trend for 5G services to shift from traditional base stations to data centers. To solve the deployment problem of 5G workloads, we use the open-source OpenLTE for performance analysis as its representative benchmark. Since open-source OpenLTE has poor performance, and it cannot reflect real scenarios and behavioral characteristics in performance analysis, we first optimize its codes according to the characteristics of general-purpose processors (GPPs) and achieve the speedup of 2.5x. On this basis, the workload execution behavior is analyzed considering the characteristics of GPPs, and the analysis shows that in the physical layer, the processing flow of the 5G downlink process is computationally intensive with a maximum port utilization rate of up to 90%, and memory access is not intensive with extremely short program response time. Finally, we analyze the workload behavior in concurrent scenarios in combination with the unique parameters of GPPs (multi-core, SMT, and turbo boost, etc.) and put forward deployment suggestions for improving data center resources. It is found that 5G workloads can only run in an exclusive machine mode on account of their strong real-time nature, and there is little competition among internal concurrent bodies for shared cache and memory access bandwidth but fierce competition for execution components. Thus, the concurrency

① 收稿时间: 2021-08-26; 修改时间: 2021-09-26; 采用时间: 2021-10-09; csa 在线出版时间: 2022-03-11

value should not exceed the number of processor cores for deployment, and the impact of TurboBoost cannot be ignored.
Key words: multi-core processor; 5G; OpenLTE; performance analysis; program optimization

数据中心将在未来迎来 5G 业务[1], 其对响应时间的要求无比严格, 通常要求精确到微秒级[2]. 人们对未来 5G 的期望远远高于现有的 4G 网络. 一方面, 随着移动互联网和物联网的发展, 人们对用户体验提出了更高的要求. 另一方面, 随着 5G 在工业互联网、车联网、企业网的深入应用, 许多场景对通信时延、通信的可靠性和安全性提出了新的要求[3]. 另外, 运营商希望以更好的整体持有成本来应对 5G 带来的大带宽, 海量流量, 多样性等特征, 其核心在于计算系统能够有更强的处理能力, 包括峰值能力和对多样业务的适应能力.

基于专用处理器的传统基站具有覆盖范围窄, 难以增加频谱效率以及软硬件方案固化等问题, 不能适应 5G 时代对通信的强实时、大带宽和升级换代便利等要求. 而数据中心采用云化的处理架构, 并将专用处理器替换为通用处理器, 可以使所有虚拟基站共享各项资源, 处理资源可以动态调度以适应不同场合, 通用处理器近年来在处理能力以及功耗方面的技术进展巨大, 且具有很好的灵活性, 能够应对各种复杂业务[4]. 所以, 出于处理能力、灵活性、多样性以及费用的考虑, 5G 应用由传统基站承载转向数据中心部署成为趋势.

5G 应用在数据中心的执行行为分析存在以下两点挑战.

(1) 无开源 5G 应用作为合适的研究对象, 而针对 4G/3G 却有很多较为成熟的开源实现, 但这些应用性能较差, 不能反映真实场景和应用特征.

(2) 行为分析的复杂性. 应用单独执行行为与所处平台特征相关, 共同运行行为则与共享资源相关, 需结合种类繁多的处理器特征参数及共享资源进行分析.

多核/众核芯片在数据中心中的普及, 使得研究人员开始关注到, 共同运行的程序之间会对共享资源产生竞争, 从而导致程序执行效率降低[5]. 针对刻画应用之间对共享资源的竞争程度, 进而预测出应用在不同场景下的性能表现, 研究人员从基于统计学习和基于深度学习两个方面开展性能预测模型的构建. Subramanian 等人建立了精确性能预测模型, 用于预测由于内存竞争造成性能损失[6]; Tang 等人使用统计模型建立了内存带宽和作业性能之间的关系[7]; 研究人员还采用诸如随机森林回归模型[8]、协同过滤算法[9]、统计分析[10]、机器学习[11]、深度神经网络[12] 等方法完成从作业运行时信息到作业性能的映射. 目前研究存在的问题主要有两点: (1) 主要选取应用的某一个特征进行分析, 如内存带宽和 Cache 失效率, 但应用的特征是多方面的, 某个角度的预测可能存在不够准确的问题; (2) 研究人员的关注重点在于在线型业务, 对 5G 等高实时应用未做针对性深入研究.

本文结合多核共享缓存、处理器执行部件和复杂硬件参数, 全面分析 5G 应用的行为特征, 得出其对存储和执行部件的资源占用情况. 然后在多线程模式下, 分析多个并发体之间的性能干扰来源, 最后得出 5G 应用在数据中心的合理部署方案. 主要贡献如下.

(1) 结合体系结构特征分析应用执行行为. 5G 应用下行过程的物理层处理流程是计算密集型的, 最高端口利用率 90%, 基本无数据复用, 访存不密集, QoS (quality of service) 要求极其严格.

(2) 以性能最佳化部署为目标, 评估 5G 应用混合执行行为并给出部署建议. 5G 应用只能独占机器执行; 不超过处理器物理核数的情况下可以尽可能多部署线程; SMT 严重影响性能因此不宜采用; 同时 TurboBoost 的影响不可忽视.

(3) 选取 OpenLTE 作为 5G 应用的代表性 benchmark 进行分析, 由于开源的 OpenLTE 性能很差, 在性能分析时不能反映真实场景和行为特征, 因此本文首先根据通用处理器的特点对其代码实现进行了优化, 取得了 2.5 倍性能加速比.

1 5G 应用优化

1.1 OpenLTE 的代表性

OpenLTE[13] 实现的是 4G 基站功能, 虽然 4G 网络与 5G 网络在性能和实现关键技术方面存在差异, 但是 OpenLTE 仍可作为研究 5G 应用下行过程物理层的样本对象, 具体叙述如下.

5G 业务相比 4G 业务, 两者部署方式及实现技术的差异主要表现在 3 方面: (1) 网络架构. 5G 的网络设计相比 4G 网络采用成本高效的密集布置, 更加简化的核心网络, 支持动态的无线拓扑的网络功能虚拟化技

术[14,15]. (2) 无线空口技术的演进, 从无线协议栈的各个层次改进, 对改善 5G 业务性能具有重大帮助[16]. (3) 运营网络管理. 运营商需要提供一个高效的资源利用、灵活的资源调配、精细和智能化的网络管理平台.

综上所述, 4G 与 5G 业务的不同之处主要在于网络结构及关键协议的具体实现, 而针对本文要研究的 5G 业务下行过程的物理层, 5G 应用和 4G 应用在关键处理过程上仍保持一致, 需要完成冗余校验、编解码、调制、解调、FFT 和多天线映射等工作, 虽然具体协议版本存在差异, 但是关键操作相似, 所以 OpenLTE 可以作为研究 5G 应用基站侧下行过程物理层的参考对象.

1.2 OpenLTE 的代码特征

OpenLTE 代码主要的功能是做数据比特流的传输、分割、整合、特定格式编码、变换码率和调制等操作. 图 1 为处理流程.

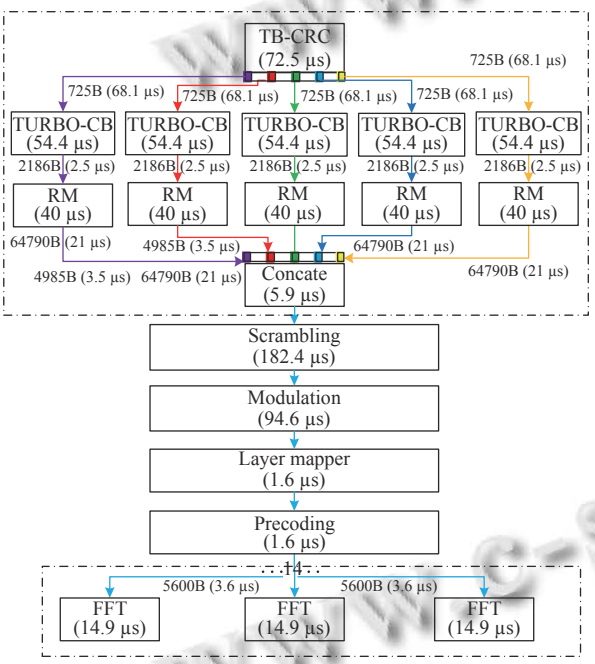

图 1 OpenLTE 程序流程图

从数据结构来说, 主要是字符数组和字符矩阵. 输入和输出数据是长串字符数组, 而在中间变换码率等操作时, 会用到矩阵来实现不同维度的编码转换.

从算法和操作的角度, 通信处理过程算法大多是位运算的复杂叠加以及数据在不同形式的数据结构和不同的传输单位间的赋值操作, 以及数据在不同坐标系间的转换操作.

1.3 OpenLTE 优化方法

本文的代码优化将从访存优化、位运算优化、分支语句优化、结合数据分布特征优化以及消除耗时运算操作 5 个方面进行[17].

1.3.1 访存优化

经分析, OpenLTE 代码中有 PreCoder, Concatenation, Layer Mapper, Rate Match, Turbo Encode 和 Segmentation 六个模块均存在大量连续的赋值操作, 且操作的数据量大, 可采用 SIMD 访存优化方法[18]. SIMD 加速部件的优势一方面体现在可以增加数据操作宽度; 另一个优势是减少了指令数, 极大提升程序的性能. 可采用 Intel Intrinsic 指令集中提供的_mm256_loadu_si256 和 store 指令进行 256 bit 的并行 load/store 操作[19].

优化后代码如代码 1 所示. 这段操作中数组是 char 型, SIMD 指令可同时进行 32 个数的 load/store, 由于数据在内存和高速缓存是连续存放的, 可以利用一个缓存行数据的空间局部性, 同时指令数大大减少, 实测加速比为 12.5 倍, 指令数减少为原来的 1/27.

代码 1. Concatenation SIMD 优化-访存宽度增加

```
//before optimization
while r < N_codeblocks do
  for j:= 0 to col step 1 do
    f_bits[j] := e_bits[j]
//after optimization
while r < N_codeblocks do
  for j:= 0 to col step 32 do
    ymm0 := _mm256_loadu_si256((__m256i *)(e_bits ))
    _mm256_storeu_si256((_m256i *)&f_bits[k], ymm0)
    k := k + 32;
```

1.3.2 位运算并行化

通信信息处理过程中的算法很多都是位运算的操作, 如 CRC 校验算法, 挖掘其背后的并行性对提升程序性能意义重大. 对 Scramble 和 Generate PRS 模块可进行位运算并行化优化, 下面以 Generate PRS 为例说明. 优化前代码如代码 2 所示.

代码 2. GeneratePRS 位运算部分代码

```
// Generate c
for i:= 0 to len step 1 do
  //x1, x2 分别进行低 3 位异或, 有效结果为最后一个 bit
  new_bit1 := ((x1 >> 3) ^ x1) & 0x1
  new_bit2 := ((x2 >> 3) ^ (x2 >> 2) ^ (x2 >> 1) ^ x2) & 0x1
  //x1, x2 分别进行最高位数值更新
  x1 := (x1 >> 1) | (new_bit1 << 30)
  x2 := (x2 >> 1) | (new_bit2 << 30)
  c[i] := new_bit1 ^ new_bit2
```

这个模块主要以移位和异或操作为主. 要对这部

分操作进行并行优化, 需要解决两个问题: (1) 解决数据依赖. 存在数据依赖意味着每次迭代操作必须串行进行, 需要消除依赖, 实现并行操作; (2) 逐位异或操作. 一般的异或操作主要是针对两个独立的变量的, 需实现位间高效位运算的方法.

优化分析: (1) 数据依赖. 真依赖主要存在于 $x1/x2$ 两个变量, 但进一步分析发现, 前一次迭代写的是 $x2$ 的最高位, 而后一次则读的是最后 4 位, 实际上并不是真依赖. 本文针对读写部分, 分别做并行化. (2) 一个数字内部逐位的异或操作, 采用数字移位并且对应位异或的方式实现. 优化后的代码如代码 3 所示. 由于函数返回的结果是 *res* 每一个位上的值, 所以需要把 *res* 逐位右移以取得最终结果.

代码 3. Generate 消除数据依赖及并行化代码

```
// Generate c
for i := 0 to col step 28 do
  //一次循环计算 28 个有效 bit 数据, 先计算 x1
  x13 := xl >> 3
  tmpl := xl3 ^ xl
  //此处和 0X7FFFFFFF 做 “与”操作, 而不是 0x1
  xl := (tmpl << 3 | xl >> 28) & 0X7FFFFFFF
  //计算 x2
  x21 := x2 >> 1
  x22 := x2 >> 2
  x23 := x2 >> 3
  tmp2 := x2 ^ x21 ^ x22 ^ x23
  x2 := (tmp2 << 3 | x2 >> 28) & 0X7FFFFFFF
  res := tmpl ^ tmp2
  //逐 bit 给 C 数组赋值
  for j := i to (28 + i) step 1 do
    c[j] := (res >> j-i) & 0x1
```

该模块基本都是位运算操作, 指令数直接决定了执行时间, 优化之后指令数变为原来的 1/2, 性能加速比为 2.45 倍.

1.3.3 分支归零化

分支语句会带来严重的性能问题. 当前 CPU 都有分支预测单元, 如果预测正确, 则条件语句只花费一个 CPU 周期, 而如果预测错误, 将可能花费几十个或更多 CPU 周期. 所以, 尽可能减少分支语句, 可以加速程序的执行时间. 本文对 Rate Match 和 Modulation 模块进行了分支优化, 以下详细介绍 Modulation 优化方法, 代码见代码 4.

代码 4. Modulation 64 种情况的 switch 语句

```
for i := 0 to N_bits step 1 do
  switch bits[i] of
  case 0:
    d_re[i] = +3 * 42
    d_im[i] = +3 * 42
    break
  case 1:
    d_re[i] = +3 * 42
    d_im[i] = +1 * 42
    break
    …
  //省略 case2, …, case63 的代码, 均与 case1 类似
  case 63:
    d_re[i] = −7 * 42
    d_im[i] = −7 * 42
    break
```

图中 switch 语句处于循环体的内部, 但不能简单地将分支语句提出循环外, 原因在于: (1) 条件判断变量与迭代变量密切相关; (2) 条件判断的计算结果完全是随机的, 分支预测失败的可能性很高. 所以这部分语句很耗时, 本文采用了查表法来代替分支选择, 将每一条分支语句都转化成查表语句.

优化分析: 该部分代码中有 64 个 case, 但经分析发现, 虽然 case 数目众多, 但是每个 case 内都进行的是赋值操作, 并且都是用同样的常数赋值, 只有符号位和系数不同, 所以采用查表法进行优化.

优化后代码如代码 5 所示. 将每个 case 的系数和符号分别实现为系数索引表和符号索引表, switch 部分的判断操作单独计算, 用计算值作为 key 索引赋值常数的系数和符号表. 经过优化完全消除了 switch 语句, 实测执行时间加速比 2.39 倍.

代码 5. Modulation 查表操作优化代码

```
Create symbol array sym_re[0..63] and sym_im[0..63]
Create coefficient array con_re[0..63] and con_im[0..63]
for i := 0 to N_bits step 1 do
  //按照索引值, 从符号表和系数表取对应数值
  index := bits[i]
  d_re[i] := sym_re[k] * con_re[k] * 42
  d_im[i] := sym_im[k] * con_im[k] * 42
```

1.3.4 特征提取优化

除了广为人知的一些优化手段外, 还可以结合程序的具体特征来做优化. Rate Match 模块优化前执行时间占到总时间的 60%. 其性能瓶颈在于: 赋值操作, 每次赋值之前都有一个条件判断, 且判断条件与迭代变量相关, 该赋值部分代码如代码 6 所示.

代码 6. Rate Match 耗时长语句

```
while k < N_e_bits do
  if(rmt_w[(k_0 + j) % N_cb] != TX_NULL_BIT)
    then e_bits[k++] := rmt_w[(k_0 + j) % N_cb]
  j:= j+1
```

优化分析: 在该段循环每执行一次赋值操作, 就要做一次 if 分支判断, 判断 *rmt_w* 的值是否为 TX_NULL_BIT. 进一步对 *rmt_w* 数组进行分析发现, 该数组中最多有 31 个值为 TX_NULL_BIT, 而整个数组的长度为将近 7 万, 所以特殊值的比例极小, 对每个数都检查一遍开销很大. 所以本文提出先并行赋值 32 个数, 然后对这 32 个数统一进行特殊值检查, 如果包含特殊值, 再逐个进行赋值, 否则, 就赋值成功. 优化后的代码如代码 7 所示.

代码 7. Rate Match 按照数据分布特征优化后代码

```
//申请 256 字节空间, 初始化为特殊值
mask := _mm256_setl_epi8(100)
while N_e_bits > 32 do
  //先一次并行拷贝 32 个数据
  src := _mm256_loadu_si256((__m256i*) & (rmt_w[(k_0 + j) %
N_cb]))
  //再检查是否存在特殊值
  res := _mm256_testz_si256(mask, src)
  //如果不存在特殊值
  if(res == 1)
  then //并行赋值 32 个数
    _mm256_storeu_si256((__m256i *) & (e_bits[k]), src)
    k := k+32; N_e_bits -=32
  else //如果存在特殊值, 逐个检查并拷贝
    for i := 0 to 32 step 1 do
      if(rmt_w[(k_0 + j) % N_cb] != TX_NULL_BIT)
        then e_bits[k++] := rmt_w[(k_0 + j) % N_cb]
    j += 32
//剩余不足 32 个数的部分, 按照原方式拷贝
```

该部分优化采用并行赋值操作, 有效减少分支语句, 实际执行指令数大幅减少, 为原来的 1/12, 实际执行时间加速比为 14 倍.

1.3.5 延迟加速化

Rate Match 的另一个性能瓶颈是高延迟操作, 该模块中出现了除法和取模操作 (取模操作用到除法部件), 本实验机器平台除法指令延迟为 89 个时钟周期, 而普通的加、减和位运算指令都是 1 个时钟周期[20], 所以非常耗时. 该部分源代码如代码 8 所示.

代码 8. Rate Match 取模和除法操作代码

```
K pi := R tc sb * C tc sb
for i := 0 to K_pi step 1 do
  pi_idx := (IC_PERM_TC[i / R_tc_sb] + C_tc_sb * (i % R_tc_sb) + 1)
% K_pi
  rmt_w[K_pi + (2 * w_idx) + 1] := rmt_y[pi_idx]
```

优化分析: 在上述代码中有两处取模操作, 一次除法操作. (1) 第 2 个取模操作只发生迭代变量达到最大的时候, 也就是只有一次计算有用, 所以可以分成 0–(*K_pi*–1), 和 *K_pi* 两种情况. 前一种情况完全省去了取模操作; (2) 除法操作和第 1 个取模操作. 可以用两层循环来代替, 内层循环的上界为 *R_tc_sb*, 内层迭代变量 *j* 表示 *i*%*R_tc_sb*, 外层循环的上界为 *K_pi*/*R_tc_sb*–1, 外层迭代变量 *k* 表示 *i*/*R_tc_sb*. 这样剩下 *R_tc_sb* 个数据尚未处理, 再分成两种情况: (1) 0 到 (*R_tc_sb*–1) 个数, 可以省去除法和取模操作; (2) 最后一个数, 也是 *i* 等于 *K_pi* 时, 直接将 *index* 为 0 时的值作为源数据拷贝. 优化后的代码如代码 9 所示. 该优化有效去除了除法操作和取模操作, 实测取得了 8 倍性能加速比.

代码 9. 消除取模操作后两层循环代码

```
K pi := R tc sb * C tc sb
//取模和除法操作分别由内外层循环变量替代
for j:=0 to (C_tc_sb-2) step 1 do
  for i:=0 to (R_tc_sb-2) step 1 do
    pi_idx := IC_PERM_TC[j] + C_tc_sb*i + 1
    rmt_w[K_pi + (2 * w_idx) + 1] := rmt_y[pi_idx]
    w_idx ++
//处理外层循环变量的边界值
j := C_tc_sb-1
for i:=0 to (R_tc_sb-2) step 1 do
    pi_idx := IC_PERM_TC[j] + C_tc_sb*i + 1
    rmt_w[K_pi + (2 * w_idx) + 1] := rmt_y[pi_idx]
    w_idx ++
//处理内外层循环迭代变量的边界值
rmt_w[K_pi + (2 * w_idx) + 1] := rmt_y[0]
```

1.4 OpenLTE 优化效果评估

本文对 OpenLTE 物理层下行过程从 TB-CRC 到 IFFT 的 10 个模块逐个进行优化, 最终整个下行过程取得 2.5 倍加速比, 模块整体最好取得 12.5 倍加速比, 被优化部分最好取得 26 倍加速比, 数据如表 1 所示.

2 5G 应用行为分析

由于通用处理器处理能力的快速发展, 5G 应用由专用处理器转移到通用处理器上. 本文将研究通用处理器上独特的参数, 如多核、SMT 和 Turbo Boost[21]

等对应用产生的影响. 通过本文前期实验验证, 当 5G 应用与其他负载混合执行时, 性能会受到不同程度的影响, 所以本文只研究 5G 应用自身多线程部署时的行为.

2.1 实验平台

本文选用的实验机器配置如表 2 所示, 测试对象为优化后的 OpenLTE 程序下行过程. 实验过程将关注该过程中每个模块 (函数) 的处理时间以及整个应用的执行时间.

表 1 OpenLTE 下行过程各模块优化效果总结

模块	优化前 (us)	优化后 (us)	加速比
Segmentation	1.01783	0.680779	1.5
Turbo Encode	3.3	2.72	1.2
Rate Match	11.05	2	5.5
Concatenation	0.743385	0.0590482	12.6
Generate PRS	3.88978	1.58408	2.5
Scramble	0.287503	0.239587	1.2
Modulation	2.24532	0.936376	2.4×
Layer Mapper	0.088088	0.016322	1
PreCoder	0.0870271	0.015978698	1
FFT	4.79	2.08321	2.3
所有模块	28.2243781	11.0608259	2.5

表 2 机器参数描述 (E5-2620)

组件	描述	
处理器	型号	Intel Xeon CPU E5-2620 Sandy Bridge
	核心数目	6核
	处理器频率	2.0 GHz (turbo up to 2.5 GHz)
	线程数目	6 (不开SMT), 12 (打开SMT)
	L1 cache	32 KB 指令缓存, 32 KB数据缓存
	L2 cache	256 KB
	L3 cache(LLC)	15 MB
内存	16 GB DDR3-1333最大带宽21.3 GB/s	

2.2 共享缓存和访存带宽竞争分析

[SMT = OFF, Turbo Boost = OFF] 本文将通过改变并发线程数的方式来研究应用对共享缓存和访存带宽的竞争情况. 因为处理器支持 SMT, 所以为了避免 SMT 对结果的影响, 本文只在一个核上部署一个线程; 同样为排除 Turbo Boost 对程序性能的影响, 关闭 Turbo Boost.

在该场景中, 单个线程可以完成 5G 应用下行过程物理层的所有处理过程, 增加线程数是提升了系统的整体并发量, 而不是将单个任务拆分为多个可并发的子任务, 该实验的目的是分析多个并发的独自处理过程之间的性能干扰情况.

如图 2 所示, 横轴表示下行过程中的不同模块, 纵轴表示执行时间. 图中每个模块从左到右分别表示并发线程数从 1 到 6. 可以看到线程的并发量逐渐增长到与核数相同, 单个线程的执行时间保持不变.

本文进一步测量程序在共享缓存的失效率和访存带宽的需求情况. 如图 3 所示, 随着在同一个 CPU 上并发的线程数不断增多, 总访存带宽逐渐增多, 但是最多到 0.3 GB/s, 而本实验机器的访存峰值带宽为 21.3 GB/s, 可见不同线程之间对访存带宽的竞争很不激烈. 并且共享缓存的失效率一直很低并且基本不变, 说明应用对共享缓存的需求很小, 即使存在多个线程, 彼此也不会争抢共享缓存.

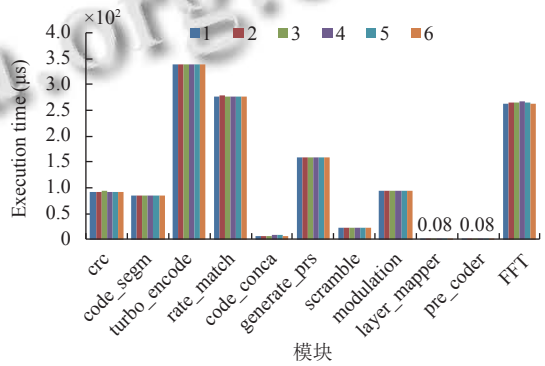

图 2 不同线程并发量时各模块执行时间

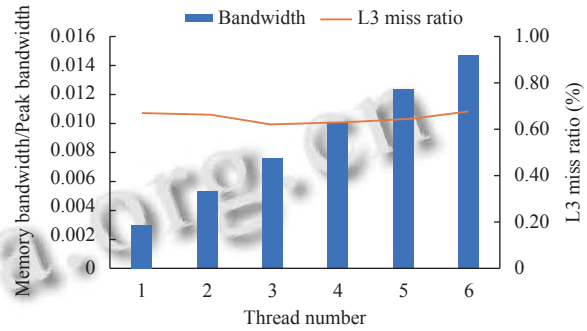

图 3 不同并发线程数总访存指标的变化

总结: 该应用的每个线程对共享缓存的需求较小, 访存量很少, 不是访存密集型的应用.

2.3 计算部件和私有缓存竞争分析

[SMT = ON, Turbo Boost = OFF] 本节将讨论由单个处理器核心支持的多线程对核内共享资源的竞争情况. 本文将对比采用 SMT 技术和不采用 SMT 技术的实验结果, 由于本文的研究关注点在一个核心内部的资源, 所以只在一个处理器核心活跃的情况下实验, 为避免 Turbo Boost 带来的干扰, 同样会关闭 Turbo Boost. 不采用 SMT 的方式, 是两个线程分别运行在两个不同的物理核上, 称为 Mapping 1; 采用 SMT 的方式, 是两

个线程都运行在同一个物理核心上, 称为 Mapping 2.

如图 4, 蓝色的柱子表示一个物理核运行一个线程, 红色柱子表示一个物理核运行两个线程, 可以明显发现当两个线程同时运行在一个物理核上时, 单个线程的执行时间明显变长, 整个过程执行时间平均增加 60%, Scramble 模块执行时间增加了一倍, 5G 应用的 SMT 线程之间存在明显的性能干扰.

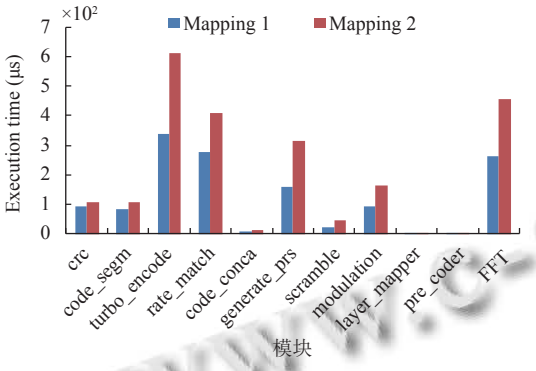

图 4 SMT 部署对单线程执行时间的影响

接下来将介绍性能干扰的原因. 测量 5G 应用在执行部件上的行为, 需要测量线程总执行周期数和处于阻塞状态的时钟周期数, 这两个指标可以反映程序执行期间执行部件的繁忙程度, 若处于阻塞状态所占时钟周期数多, 则代表程序大部分时间都在等待. 本文对单线程和 SMT 情况下分别统计 UOPS_RETIRED.TOTAL_CYCLES/UOPS_RETIRED.STALL_CYCLES, 这两个指标分别表示该进程的微指令无论处于阻塞或者执行状态的所有的时钟周期即进程的总时钟周期数, 后一个指标表示微指令处于阻塞状态的时钟周期数.

如图 5 所示, 只有单线程执行时, 处于阻塞状态的时钟周期数占到总的时钟周期数的 20%, 表明 5G 应用是计算较密集的应用. 当采用 SMT 时, 整个应用的时钟周期数明显增加, 对比两个柱子可以看到, 执行时间并未有变化, 增加的时钟周期数全部来源于阻塞时钟周期数. 指令被阻塞可能的原因是来自于执行部件的竞争, 或者等待来自于缓存的数据. 所以本文将 SMT 状态下的阻塞时钟周期数做了进一步分析, 发现分为四部分: 来自于单线程执行状态下已有的阻塞、由于等待 L1 缓存中的数据造成的阻塞、由于等待 L2 缓存中的数据造成的阻塞以及由于另一个线程占据执行部件而造成的阻塞, 而大部分阻塞都是由于等待另一个线程释放执行部件造成的, 证明 5G 应用 SMT 线程间对执行部件的竞争激烈.

上述实验本文论证了 5G 应用在采用 SMT 方式带来的负面作用, 接下来将从处理器资源利用率的角度来说明 SMT 的好处.

本文从执行部件的利用率和吞吐量两个角度来说明. 如图 6 所示, 在本实验机型中, 端口 0、1 和 5 表示 ALU 等计算部件, 而端口 2、3 和 4 则表示访存部件, 对于 5G 应用, 计算部件的利用率明显高于访存部件, 再次说明这是计算密集型应用, 而蓝色柱子表示单线程执行时的部件利用率, 而红色柱子表示 SMT 线程执行时的部件利用率, 采用 SMT 之后, 各部件的利用率都提升.

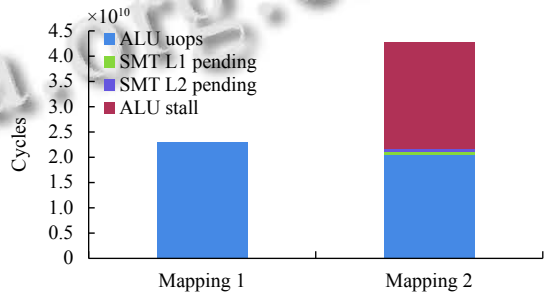

图 5 采用 SMT 部署执行和阻塞部分时钟周期数

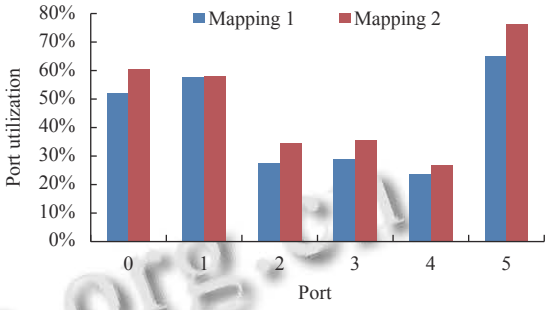

图 6 采用 SMT 部署各部件的利用率

如图 7 所示, 蓝色表示单线程执行时, 大部分都是每个时钟周期执行 1 条或 2 条微指令, 而红色采用 SMT 之后, 每个时钟周期能同时执行 3 条或者 4 条微指令的比例大幅增加, 大多数时钟周期都能执行 1、2 或 3 条微指令, 提升了处理器的吞吐量.

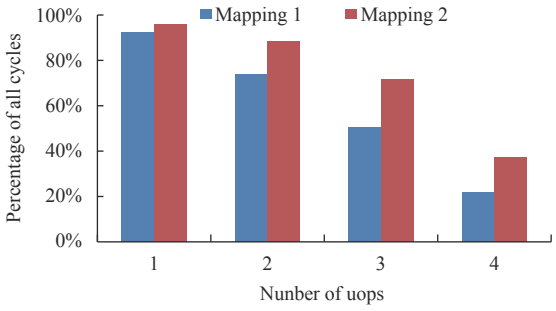

图 7 采用 SMT 部署吞吐量的变化

总结: 5G 应用计算很密集, 执行部件的利用率较高, 不采用 SMT 方式时, 5G 应用可以占满所有处理器核心; 采用 SMT 方式时, 线程之间性能干扰严重.

2.4 Turbo Boost 影响分析

[SMT = OFF, Turbo Boost = ON] 本节将讨论即使不存在资源竞争时, 处理器本身的特性——超频, 对应用并发执行时的性能影响, 并且 SMT 将关闭. 超频对处理器主频的影响如表 3 所示.

如图 8, 应用执行时并发的线程数量从 1 增加到 6, 1 或 2 个线程并发执行时, 每个线程的执行时间最少; 3 或 4 个线程并发执行时, 每个线程的执行时间增加 5%; 5 或 6 个线程并发执行时, 每个线程的执行时间相比单线程执行时增加 10%. 所以, 虽然线程间对共享资源的竞争不激烈, 由于超频的作用, 当活跃处理器核心数增多, 每个线程的响应时间增加了.

表 3 Turbo Boost 频率变化

活跃处理器核心数	处理器频率 (GHz)
基本频率	2.0
1或2	2.5
3或4	2.4
5或6	2.3

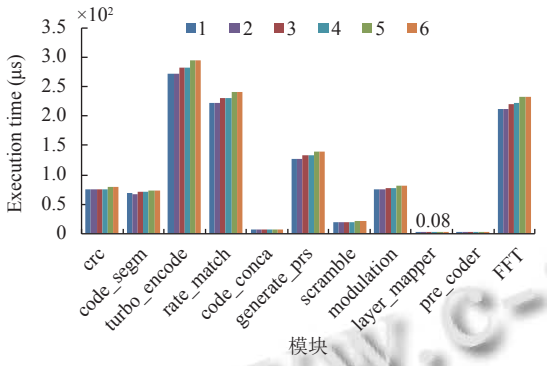

图 8 开启 Turbo Boost 时, 单线程执行时间随活跃处理器核心数目变化的规律

本文把打开超频和不打开超频的结果做个对比, 如图 9, 前 3 种颜色的柱子表示打开超频, 并行活跃的处理器核心数为 1、3 和 5 三种情况, 后一种颜色的柱子表示关闭 Turbo Boost 的结果, 可以看到关闭 Turbo Boost 之后, 执行时间明显大幅增加, 约增加 20%.

结论. 超频可以显著提升应用的性能, 但是同时需要注意的问题是, 实际软件开发过程中, 往往测试应用的响应时间会从单线程开始, 在对应用和设备足够了解的情况下, 即使能断定多线程执行时, 并发体之间不会由于资源竞争产生性能干扰, 但由于 Turbo Boost 的影响, 多线程执行时, 每个线程响应时间也会有一定增加, 所以对单线程性能基准的评定应提高要求, 这样才能保证应用在多线程并发时, 仍然满足性能要求.

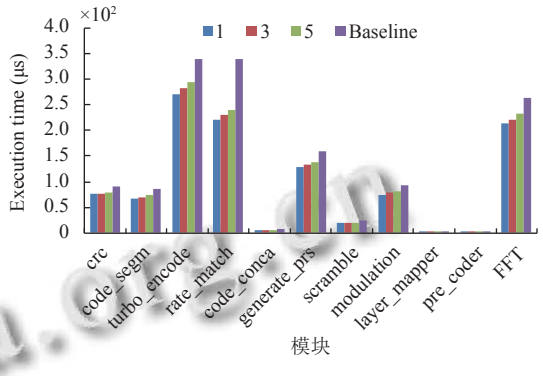

图 9 Turbo Boost 对性能的影响程度

2.5 5G 应用在数据中心部署建议

通过对 5G 应用下行过程物理层全面的分析评估, 对应用自身特性和并发执行的行为得出的结论如下:

(1) 应用特性. 5G 应用下行过程物理层数据解析过程是计算密集型的, 对于共享缓存的压力较小, 实际占用的访存带宽也很小, 对响应时间要求极高, 处于微妙级.

(2) 混合执行行为及部署建议. 1) 不采用 SMT, 由于该应用是计算密集型的, 执行部件的利用率已经很高, 两个线程同时执行时, 单个线程被阻塞的时间增加, 性能下降了. 2) 线程并发量. 多核多线程执行时对共享资源竞争不激烈, 但是同一个核心多线程之间由于竞争执行部件性能干扰严重, 所以在不超过核数时尽量可以多部署线程. 3) Turbo Boost, 可以选择打开或不打开, 各有利弊. 打开 Turbo Boost 时, 程序的性能相对较好, 但是性能会随着活跃处理器核心数目变化而不稳定, 影响单线程性评测时的基准的设定. 不打开 Turbo Boost 时, 性能不会随着活跃处理器核心数变化而抖动, 但是整体性能一般.

3 结束语

本文结合体系结构特征分析 5G 应用执行行为, 发现 5G 应用下行过程的物理层处理流程是计算密集型的, 最高端口利用率 90%, 然后分析应用并发行为, 最后以性能最佳化为目的给出 5G 应用在数据中心部署建议, 应用对实时性要求极其严格, 只能应用独占机器执行; 计算工作集小, 对内存及其子系统压力小, 不超过处理器物理核数的情况下可以尽可能多部署线程;

SMT 会造成响应时间增加 60%, 严重影响性能因此不宜采用. 本文还针对开源实现的 OpenLTE 代码进行面向通用处理器特征的优化, 设计并实现了访存并行化、位操作并行化、分支归零化、特征提取优化和延迟加速化等优化方式, 整体优化后加速比为 2.5 倍, 单个优化基本块取得最好性能加速比为 26 倍. 本文仅针对 5G 应用下行过程的物理层进行特征刻画, 还不能够全面评估应用整体的行为, 后续将会继续对物理层上行链路进行分析, 从而更加准确分析应用在数据中心的行为, 以达到资源的合理部署, 提升数据中心资源利用率.

参考文献

1 肖子玉. 面向 2030 的未来网络关键技术综述——Beyond 5G. 电信科学, 2020, 36(9): 114–121.

2 杨旭, 肖子玉, 张明, 等. 5G 核心网面向 3GPP R16 演进关键技术及引入策略. 电信科学, 2021, 37(6): 150–159.

3 Liu Y, Wang CX, Huang J. Recent developments and future challenges in channel measurements and models for 5G and beyond high-speed train communication systems. IEEE Communications Magazine, 2019, 57(9): 50–56. [doi: 10.1109/MCOM.001.1800987]

4 Nasirian S, Faghani F. Crystal: A scalable and fault-tolerant Archimedean-based server-centric cloud data center network architecture. Computer Communications, 2019, 147: 159–179. [doi: 10.1016/j.comcom.2019.08.004]

5 Zhu HS, Lo D, Cheng LQ, *et al*. Kelp: QoS for accelerated machine learning systems. Proceedings of 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). Washington, DC: IEEE, 2019. 172–184.

6 Subramanian L, Seshadri V, Kim Y, *et al*. Predictable performance and fairness through accurate slowdown estimation in shared main memory systems. arXiv: 1805.05926, 2018.

7 Tang XC, Wang HJ, Ma XS, *et al*. Spread-n-share: Improving application performance and cluster throughput with resource-aware job placement. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Denver: ACM, 2019. 12.

8 Barve YD, Shekhar S, Chhokra A, *et al*. FECBench: A holistic interference-aware approach for application performance modeling. Proceedings of 2019 IEEE International Conference on Cloud Engineering (IC2E). Prague: IEEE, 2019. 211–221.

9 Romero F, Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems. Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. Limassol: ACM, 2018. 19.

10 Arndt OJ, Lüders M, Riggers C, *et al*. Correction to: Multicore performance prediction with MPET: Using scalability characteristics for statistical cross-architecture prediction. Journal of Signal Processing Systems, 2021, 93(11): 1361. [doi: 10.1007/s11265-021-01691-x]

11 Witt C, Bux M, Gusew W, *et al*. Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. Information Systems, 2019, 82: 33–52. [doi: 10.1016/j.is.2019.01.006]

12 Gan Y, Zhang YQ, Hu K, *et al*. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. Providence: ACM, 2019. 19–33.

13 Wojtowicz B. OpenLTE. An open source 3GPP LTE implementation. https://sourceforge.net/projects/openlte.

14 Adachi F, Takahashi R, Matsuo H. Challenges for beyond 5G: Ultra-densification of radio access network. Journal on Communications, 2020, 41(11): 1–11.

15 Kaljic E, Maric A, Njemcevic P, *et al*. A survey on data plane flexibility and programmability in software-defined networking. IEEE Access, 2019, 7: 47804–47840. [doi: 10.1109/ACCESS.2019.2910140]

16 Chen Z, Ma XY, Zhang B, *et al*. A survey on terahertz communications. China Communications, 2019, 16(2): 1–35.

17 Falk H, Marwedel P. Source code optimization techniques for data flow dominated embedded software. Boston: Springer Science & Business Media, 2004.

18 Estérie P, Gaunard M, Falcou J, *et al*. Boost. SIMD: Generic programming for portable simdization. Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT). Minneapolis: IEEE, 2012. 431–432.

19 Inter intrinsics guide. http://software.intel.com/sites/landingpage/IntrinsicsGuide.

20 Granlund T. Instruction latencies and throughput for AMD and Intel x86 processors. 2008.

21 Charles J, Jassi P, Ananth NS, *et al*. Evaluation of the Intel® Core™ i7 turbo boost feature. Proceedings of 2009 IEEE International Symposium on Workload Characterization (IISWC). Austin: IEEE, 2009. 188–197.