计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191200180

基于神经网络的循环分块大小预测

池昊宇　陈长波
中国科学院大学重庆学院　重庆 400714
中国科学院大学计算机科学与技术学院　北京 100049
(chihaoyu@cigit.ac.cn)

摘　要　循环程序的优化一直是程序优化的重点,循环分块作为一种典型的循环程序优化技术已被广泛地研究和应用。分块大小的选择对循环程序的性能有着重要影响,分块大小的选择复杂多变且高度依赖程序和硬件。传统的静态分析和启发式经验搜索的人工和时间成本过高,缺少通用性和可移植性。为此,考虑使用有良好高维表示特性的神经网络方法来学习程序与硬件复杂交互过程中分块大小与程序性能的隐含关联。从问题规模、循环结构、循环内操作的局部性等方面抽取出一组新的29维特征,对问题规模为1024～2048的随机大小的6类内核程序(3维循环、2维数据)的数十万行示例进行实验。串行模型(TSS-T6)相比GCC-O2默认优化实现了6.64倍的平均加速比,相比穷尽搜索实现了98.5%的平均最大可用性能,相比Pluto默认分块优化实现了平均9.9%的性能提升。并行模型(TSSP-T6-Search)相比OpenMP默认优化实现了2.41倍的平均加速比,相比穷尽搜索实现了91.7%的平均最大可用性能,同时与Pluto默认分块并行优化相比得到了平均9%的性能提升。

关键词:编译优化;自动调优;循环程序分块;人工神经网络;缓存优化
中图法分类号　TP314

Prediction of Loop Tiling Size Based on Neural Network

CHI Hao-yu and CHEN Chang-bo
Chongqing School,University of Chinese Academy of Sciences,Chongqing 400714,China
School of Computer Science and Technology,University of Chinese Academy of Sciences,Beijing 100049,China

Abstract　Optimization of loop programs is an important topic in program optimization. As a typical technique of loop optimization,loop tiling has been widely studied and applied. The selection of tile size,which is complex and highly dependent on program and hardware,has a great impact on the performance of loops. Traditional approaches based on static analysis and heuristic experience search have high cost of labor and time,and lack generality and portability. Therefore,the neural network method,which is good at representing high-dimensional information,is considered to learn the hidden correlation between tiling size and program performance,which is a result of complex interaction between program and hardware. A new group of features with 29 dimensions are extracted based on size of the problem,structure of the loop,locality of the operations within the loop. The experiments are carried out on hundreds of thousands of lines of six kinds of kernel programs (3D loop,2D data) with random sizes in the range of 1024 to 2048. The sequential model (TSS-T6) achieves an average speedup of 6.64 compared with GCC-O2,98.5% of the average maximum available performance compared with the exhaustive search,and an average 9.9% performance improvement compared with Pluto. The parallel model (TSSP-T6-Search) achieves an average speedup of 2.41 compared with the OpenMP default optimization,91.7% of the average maximum available performance compared with the exhaustive search,and an average 9% performance improvement compared with Pluto default parallel tiling optimization.

Keywords　Compilation optimization,Automatic tuning,Loop tiling,Artificial neural network,Cache optimization

1　引言

在科学计算中,一些重要应用程序的计算,尤其是数据密集型程序的内核是循环的,相对于整个程序的运行时间,这些循环的开销最大[1],优化循环可以提升整个程序的性能。为了优化循环,需要充分理解程序在执行过程中的行为特征,如

到稿日期:2019-12-30　返修日期:2020-05-20　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金面上项目(11771421,11671377,61572024);中国科学院"西部之光";重庆市院士牵头科技创新引导专项(cstc2018jcyj-yszxX0002)
This work was supported by the National Natural Science Foundation of China (11771421,11671377,61572024),Chinese Academy of Sciences "Light of the West",Chongqing Academician-led Science and Technology Innovation Guidance Project (cstc2018jcyj-yszxX0002).
通信作者:陈长波(chenchangbo@cigit.ac.cn)

缓存行为、硬件数据预取行为、局部性原理等。其目的是增加数据重用，减少缓存冲突和减小未命中率，充分利用时间局部性和空间局部性对循环程序进行优化。近年来，不少研究将程序优化作为编译器优化的一部分，利用机器学习方法从最佳参数因子选择、最佳编译选项组合和排序等方面进行研究[2-4]，如循环分块[5-7]、循环展开[7-8]、线程粗化[9]和映射[10-12]等。

循环分块作为循环程序的一种经典的优化技术被广泛地研究和应用[7]，分块技术根据分块大小和形状对迭代空间进行重新排序，然后进行计算，以减少数据重用距离，最小化缓存未命中率，增强在更高级别内存层次结构中数据访问的友好性，实现粗粒度并行。分块大小受到缓存行大小、关联度、数据替换策略、硬件存储架构等多种因素的共同影响。当分块过大时，数据无法充分利用缓存或寄存器空间进行重用，就有可能被置换出去，从而导致缓存未命中的情况频繁发生；当分块过小时，可能会造成较大的调度成本，无法充分发挥分块带来的优势，最优和最差的分块大小可能导致性能相差数倍。当涉及多层循环嵌套时，如果要对每层循环选择一个合适的分块大小，就要考虑每层循环之间的各种数据依赖关系[5]。

分块大小的选择策略可分为基于程序和硬件的静态分析、基于样本空间的经验搜索和基于程序分块特征的机器学习模型预测这三大类。基于程序和硬件的静态分析方法主要通过分析访存过程中循环体数据的依赖关系，来综合考虑分块执行的数据重用情况、分块中使用的数据在内存和缓存中的布局[13-15]、缓存未命中和命中情况的边界、缓存替换策略、容量未命中和冲突未命中的避免[16-17]、硬件的预取、数据的向量化情况等因素，建立解析模型来指导分块的选择。静态分析方法在理论假设和约束下分析分块过程各个方面的复杂度，在减小缓存未命中率、降低各种冲突失效、提升数据重用、充分利用时间和空间局部性方面有较好的表现，但是这类理论且静态的分析方法在实际应用中仍然无法全面地考虑到更多潜在的细节，无法精确模拟实际的复杂过程，建模过程相当繁琐，且全靠手工来分析，成本代价过高，严重依赖硬件和程序布局[5]。基于样本空间的经验搜索将静态分析模型与经验搜索相结合来执行启发式的搜索，通过静态分析模型构造出指导样本空间迭代搜索的成本函数；然后考虑全局空间的特征，在有效的修剪搜索空间中进行局部搜索，可以有效地缩减搜索代价[18-20]。基于空间搜索的方案通过实际尝试不同的参数来进行多次迭代编译选优，往往可以获得优于静态分析方法的性能，但是当面临多层循环和较大规模的数据时，要遍历这种组合爆炸式的搜索空间，其时间成本仍让人望而却步[21-22]。

基于机器学习的模型预测方法将循环分块问题看作在特定计算机系统下，硬件与数据的复杂交互-决策过程（相关性）。目前，研究人员常采用神经网络的方法进行研究[6,23-26]。例如，文献[6]利用神经网络选择最佳循环分块因子，从 9 个程序内核中捕捉循环分块的时间和空间局部性特征，手工选取 6 个特征，通过全连接神经网络模型实现了 80%～95%的最大可用性能。文献[23]利用广义回归神经网络（General Regression Neural Network，GRNN）选择循环最佳分块因子，从 PolyBench 和 BLAS 基准程序测试套件中选取 20 个循环程序作为原始程序数据集；然后由人工来提取数据集的特征，将包含问题规模（ProblemSize，PS）、最内层矢量化特征 V 以及各层数据访问的时间局部性和空间局部性一同作为程序特征，实现了平均 90%的最大可用性能。文献[24]使用人工神经网络来预测给定基准的分块大小的性能分布。通过针对 7 个基准的实验表明，该方法在识别分块大小方面非常有效，从而使整个搜索空间的执行时间占全局最小执行时间的 10%以内。以上研究表明，使用神经网络方法解决 TSS 问题是有效的，且其在性能上的表现平均达到了 90%，相比静态分析和经验搜索，大大减少了人工成本和时间成本，是利用机器学习方法探索程序优化问题的有效尝试。但是，一方面，手工提取的特征是否能够充分地反映程序运行过程的特征还有待证明；另一方面，这些研究暂未充分考虑不同问题规模的 TSS 问题，且对于这些问题数据集的规模都比较局限（数千个样本），是否能在更多的数据集样本上构建一个充分考虑问题规模变化、样本数据量更大（上万或数十万）、更为精确、性能更好、更加通用的基于神经网络的 TSS 模型成为本文研究的重点。

本文提出的基于神经网络的方法用于预测特定类别程序的随机问题规模下的最佳分块大小。利用神经网络模型拟合该问题的性能曲线，实现针对随机规模程序的执行时间预测，然后根据预测的前 10%的执行时间选出候选最佳分块大小，以进一步选出最佳分块的大小。基于 TSS 的模型与相关研究的对比如下：本文考虑了比文献[6，23-24]更为丰富的特征集，在文献[24]的基础上加入了操作数特征，并结合文献[23]的局部性特征分析方法重新构建了更为详尽的局部特征集；本文针对循环结构进行分析和编码，这是以上基于神经网络的 TSS 研究都未考虑到的问题；此外，本文考虑了更为全面的问题规模下的分块大小预测，相比同类型研究的数千的数据量，本文使用的数据量达到了数十万。

2 研究背景

对于循环程序来说，分块和未分块的数据空间局部性有着巨大的差异，以典型的三层循环下的二维数组乘法基准程序 Matmul 为例，图 1(a)、图 1(b)分别展示了利用工具 Pluto[27]对基准程序 Matmul 分块前后的代码变化，用参数化的因子 I,J,K 表示从外到内的循环可能选择的分块大小。分块后的源码由之前的三层循环增加到六层循环，前三层循环用局部变量 t_1,t_2,t_3 代表对应的循环层，这是用于控制分块的块间（Intertile）循环；后三层循环用局部变量 t_4,t_5,t_6 代表控制分块的块内（Intratile）循环。值得一提的是，循环的局部变量 i,j,k 在分块后顺序被调整为 i,k,j，如图 1(b)中对应的 t_4,t_5,t_6，这是由 Pluto 通过多面体相依性分析、整数线性规划、启发式的确定循环变换，以最小化分块之间的通信开销处理后的结果。本文中所有示例程序的分块都是基于 Pluto 的分块变换策略进行的。

为了直观地展示分块和未分块的循环程序在执行时间上的差异，图 2(a)展示了未分块和分块循环程序随问题规模的执行时间的变化情况。问题规模（Problem Size，PS）为 400～4 000，间隔 100 取方阵 $\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}$ 作为 Matmul 内核参与运算的矩阵，参与运算的数据类型均为 double；分块大小（Tile Size，TS）从 100 开始以 100 为间隔取至 PS，且每层循环的分块大

小都取相同的 TS，纵轴表示循环程序的执行时间。

```
#pragma scop
    for(int i=0;i<PS1;i++){
        for(int j=0;j<PS3;j++){
            for(int k=0;k<PS2;k++){
                C[i][j] += A[i][k] * B[k][j];}}}
#pragma endscop
```

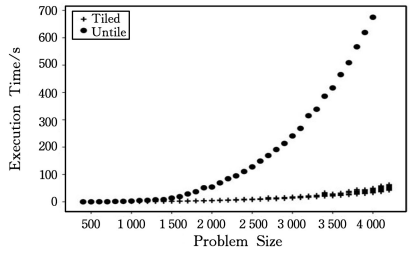
(a)分块前的源码

```
int t1,t2,t3,t4,t5,t6;
register int lbv,ubv;
/* Start of CLooG code */
if((PS1 >=1)&&(PS2 >=1)&&(PS3 >=1)) {
for(t1=0;t1<=floord(PS1-1,I);t1++) {
    for(t2=0;t2<=floord(PS3-1,J);t2++) {
        for (t3=0;t3<=floord(PS2-1,K);t3++) {
            for (t4=I*t1;t4<=min(PS1-1,I*t1+I-1);t4++){
            for (t5=K*t3;t5<=min(PS2-1,K*t3+K-1);t5++) {
                lbv=J*t2;
                ubv=min(PS3-1,J*t2+J-1);
#pragma ivdep
#pragma vector always
            for (t6=lbv;t6<=ubv;t6++) {
                C[t4][t6] +=A[t4][t5] * B[t5][t6];}}}}}}}
/* End of CLooG code */
```

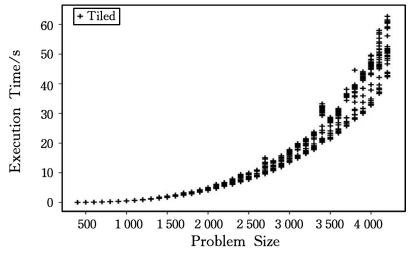
(b)经 Pluto 分块处理后的源码

图 1 基准内核 Matmul 循环部分分块前后的源码

Fig. 1 Source code of Matmul loop un-tiling and tiling with Pluto



(a)不同问题规模的循环程序未分块和分块下的执行时间对比



(b)不同问题规模的循环程序在不同分块下的执行时间分布

图 2 分块与不分块情况下的循环程序执行时间对比

Fig. 2 Comparison of execution time between tiling and un-tiling

从图 2 可以看出，随着问题规模的增大，未分块和分块程序的执行时间复杂度为 O(PS3)，分块能大幅度地缩短程序的执行时间。图2(b)单独展示了分块循环程序的执行时间随问题规模变化的情况，可以看出，随着 PS 规模的增大，不同分块间的执行时间差异也逐渐增大，说明随着问题规模的增大，不同的 TS 对程序的执行时间有越来越大的影响，因此 TS 的选择变得十分重要。对于某些反复用到的大型科学计算内核，随机或按经验地选择一个分块大小，相比性能相对最优的分块大小，随着程序规模的增大和程序的大量运行，累计的差异也是巨大的。如图 2(b)所示，对于 PS 在 4000 左右的程序，其最差分块和最优分块下单个程序的每次执行时间的差异就超过了 20s，如果能找到“最优”分块性能或避免“最差”分块性能，将带来可观的性能提升。

为了充分利用局部性原理提高数据的重用率，优化程序的执行过程，需要理解缓存(Cache)的结构和工作机制[28]。现代的高速缓存普遍采用多级缓存结构，将第 k 层的缓存 L_k(更快、更小的存储设备)作为第 $k+1$ 层的缓存 L_{k+1}(更大、更慢的存储设备)的缓存。通常 L_{k+1} 存储器被划分成连续的数据对象块(Block)，这样的传送块也被称为缓存行(Cache Line)；数据总是以块为单位在第 k 层和第 $k+1$ 层之间来回传送。以 Intel core i7 处理器[29](8 核)的缓存(见表 1)为例，L_1 与 L_2，L_2 与 L_3 之间传送的块大小都为 64 B。从表 1 可以看出，如果能够在 L_1 中缓存命中，其访问时间只需要 4 个时钟周期，在 L_2 中命中需要 10 个时钟周期，在 L_3 中命中需要 40～75 个时钟周期；如果第 k 层中没有缓存数据对象 D，那么就是缓存未命中(Cache Miss)，因未命中会使后续策略的时间代价急剧增加，如到内存中读取就要经历数百个时钟周期。分块技术可以对原工作集进行合理划分，对迭代空间重新排序和分块，以缩短数据重用距离，利用缓存结构中的数据局部性原理，减少在缓存结构中可能由于不合理的数据重用和依赖导致的反复替换和未命中行为，最小化缓存未命中，即提升缓存的命中率，进而有助于实现粗粒度的并行。实际的缓存结构更加复杂和多变，由于分块问题高度依赖硬件实现，因此对循环分块问题的分析方法也必须详细考虑各种硬件细节[30]。这不仅导致了问题的复杂性，同时也无法做到精确的分析和通用的建模，是分块问题面临的一大挑战。

表 1 多级缓存结构的参数信息

Table 1 Parameter information of multi-level cache structure

Type	Cycle time (clocks)	Size	Cache Line/B	相联度 E	组数 S
L1 i-cache	4	32 kB×8	64	8	210
L1 d-cache	4	32 kB×8	64	8	210
L2	10	256 kB×8	64	4	211
L3	40～75	12 MB	64	12	214

3 构建 TSS 模型

本节将结合静态分析模型讨论样本空间、问题特征等相关问题。使用神经网络模型来估计性能分布，必须承认无法穷尽所有的参数配置，必须通过合理的采样和剪枝，保留最具代表性的数据特征来训练模型。

TSS 模型的构建过程如下：1)评估搜索空间的一部分，收集程序和限制搜索空间；2)进行程序的特征工程，包括预处理和特征编码；3)利用穷尽搜索获得对应程序和分块大小选择的执行时间；4)构建给定程序和数据集大小的人工神经网络(ANN)；5)使用基于 ANN 模型的预测数据(执行时间)，围绕性能(执行时间)分布的预测执行时间的最小值来确定一组(10%)预测的最小执行时间，值得注意的是，不需要完全拟合整个非线性的曲线，只需要拟合好性能曲线的最低点的局部曲线就足以支撑在此基础上选择对应的分块大小；6)选择执行时间从短到长前 10%的样本所对应的候选分块集，按照随机策略或进一步搜索策略确定最佳块。实验数据收集和测试

的平台配置如下：操作系统为 64 位 Linux 4.4.0，编译器为 GCC 5.4.0，处理器为 Intel(R) Core(TM) i7-9700F CPU，主频为 3.00GHz，核心数为 8，内存为 8GB。

3.1 数据集

从 PolyBench[31] 和 BLAS 中选取的基准程序内核如表 2 所列，这些内核都是三层循环下的二维矩阵或向量运算，这些基准程序是在科学计算中常用的，同时众多的循环分块研究也都不同程度地选取了这些基准程序作为源数据，例如文献[6，23-25，31-33]等。

表 2 所选用的基准程序

Table 2 Selected benchmark procedures

Kernel	Brief description
Matmul	Matrix multiplication
Dsyrk	Symmetric rank-k update
Dsyr2k	Symmetric rank-2k update
Tmm	Triangular matrixmultiplication(three matrices)
Trmm	Triangular matrix multiplication (two matrices)
Gemm	Generalizedmatrix multiplication from BLAS

(1)串行(单核)实验数据收集。以 GCC-O2 的执行时间为基准，使用 Pluto 的-tile 进行分块代码转换，并参数化分块大小，以 GCC-O2 执行编译优化收集实验数据。

(2)并行(八核)实验数据收集。首先使用 Pluto-parallel 将串行代码转换为并行程序，然后用 OpenMP(-fopenmp，-O2)指导并行，以此作为并行条件下的基准；使用 Pluto 的-tile-parallel 进行并行化和分块，并参数化分块大小，再用 OpenMP(-fopenmp，-O2)指导并行收集实验数据。

3.2 问题规模

从图 1(b)可以看出，当 PS 较小时，分块和未分块对程序性能的影响很小，进行分块的额外成本反而淹没了提升的性能。首先考虑 PS 的单个维度为 1 024 至 2 048 的矩阵，本文与文献[6，23，24]的不同点之一就是广泛考虑了不同矩阵大小对分块问题的影响。例如，文献[6]考虑了大小为 1 000×1 000 和 3 000×3 000 左右较小范围的问题规模；文献[23]只考虑了几种特定(大小分别为 512×12，1 024×1 024，2 000×2 000，4 000×4 000)规模的数据；文献[24]只针对大小为 600×600 这一种问题规模。实验表明，矩阵大小的差异是影响分块选择的重要因素，而且并不能通过某一大小的 PS 代表其周围大小 PS 的情况。因此，本文的问题规模是在一定范围内的符合二维数组(可视为矩阵)运算规则的随机大小组合，这样的数据更符合实际情况下的优化任务。

3.3 分块大小(Tile Size，TS)

为了获得不同问题规模和分块大小组合下的数据，必须选择合理且具有代表性的分块大小样本空间。在复杂度方面，当非立方体块在三层循环问题中时，假设针对每层循环的候选分块 TS 为 N，那么对于三层循环，其分块的空间就为 N^3；此外，考虑立方体的文献一般固定矩阵大小或使其在小范围内变化，但本文为了充分考虑矩阵的大小，先限定立方体分块，再考虑非立方体分块的作用。在寻找大数据集的最佳分块大小时，收集有限范围的最佳分块是一种有效的剪枝策略[24]。

分块取值问题包括分块大小的范围和间隔的选择，具体确定思路如下：1)分块的取值应小于问题规模的正整数；2)立方体分块的间隔选择是 $CLS = cache\ line \backslash / sizeof(DataType)$ 的倍数；3)选择的立方体分块所对应的工作集应当充分利用 L2 缓存；4)避免过大分块带来的边界递减效应，即额外的重用导致了回报递减，分块的大小超过 $PS_{\min}/2$ 之后程序执行时间绝大部分都延迟了(我们在更广泛的内核和问题规模下都得到了相似的现象)。根据上述思路，分块的取值为：

$$UL = \min\left(\mathrm{sqrt}\left(\frac{L2}{\mathrm{size\ of(DataType)}}\right), PS_{\min}1/2\right) \tag{1}$$

$$TS = k * CLS, k = 1, 2, 3, \cdots, UL/CLS \tag{2}$$

其中，L_2 的大小为 256kB，DataType 为 double，缓存行大小为 64B。

3.4 循环结构特征

文章首页 OSID 码中给出了 6 类 Kernel 循环结构的差异，这些 Kernel 的主要不同点在于每层循环初始化不同。由此可以分析得出表 3 所列的结构特征：i，j，k 分别表示由最外层循环到最内层循环的局部变量，以 Matmul 为例，其由外到内的循环 i，j，k 的初始化都是 0，因此在表 3 中对应初始化位置取 1，即其结构特征在样本空间的结构特征中被编码成 110100，同理 Tmm 被编码成 101010，Dsyrk 被编码成 110001。值得注意的是，由于所有示例中第一层循环的局部变量的初值都为 0，因此从编码的角度，该特征值都为 1，故在实际的训练过程中可以不加入该项特征进行训练。

表 3 循环程序结构的特征

Table 3 Features of loop program structure

下标	i	j		k		
初始值	0	0	i	0	i	j
Matmul	1	1	0	1	0	0
Tmm	1	0	1	0	1	0
Dsyrk	1	1	0	0	0	1
Dsyr2k	1	1	0	0	0	1
Trmm	1	0	1	0	1	0
Gemm	1	1	0	1	0	0

3.5 读写过程中的局部性特征

除了考虑循环程序的循环结构特征外，还需要考虑每层循环与内循环操作之间的数据读取及预取特征。为了更好地区分分块循环维数之间的效果差异，准确地捕捉和编码程序特征，我们严格按照由内层循环到外层循环的顺序进行分析。L_1 表示第一层循环的局部性特征，L_2 表示第二层循环的局部性特征，L_3 表示第三层循环的局部性特征。受文献[6，24]的启发，进一步针对块内(Intratile)的每层循环的局部特征，依据内循环的操作按照读(Read，R)/写(Write，W)、可预取(Prefetched，P)成向量/不可预取(Non-Prefetched，NP)向量/常量式不依赖(Invariant，I)等行为，组合成关于读(R)的 RP，RNP，RI 特征和关于写(W)的 WP，WNP，WI 特征，进一步添加操作数(Operand Number，ON)特征，该特征表示内循环中参与计算的矩阵/向量的个数。针对所选 Kernel，以上特征满足如下约束：1)WP，WNP，WI 互斥；2)ON－1＝RP＋RNP＋RI。表 4 列出了部分 Kernel 的局部性特征。下面以 Matmul 为例，分析由内至外的循环。首先是块内(Intratile)循环中的最内层循环 L_3，$L_3(j)$ 表示当前循环层的局部变量对应分块前的 j 层循环，类似地，进一步分析 $L_2(k)$ 和 $L_1(i)$。值得注意的是，Matmul 分块后原始循环的 i-j-k 顺序变成了 i-k-j 的块内循环顺序，这是 Pluto 根据循环的自动调整策略。Mat-

mul 的局部性特征分析和编码过程如下。首先，对于最内层循环 L_3，由于变换的是 j，因此：1）$B[k][j]$，$C[i][j]$可读预取（重用），即 $RP=2$；2）$A[i][k]$为 RI，即 $RI=1$；3）$C[i][j]$可写预取（重用），即 $WP=1$。其次，对于第二层循环 L_2，由于此时变换的是 k，因此：1）只有 $A[i][k]$可读预取，即$RP=1$；2）$B[k][j]$无法读预取，故 $RNP=1$；3）$C[i][j]$可读写重用，不依赖 k，即 $WI=1$，$RI=1$。最后，对于最外层循环 L_1，由于此时变换的是 i，因此：1）$A[i][k]$，$C[i][j]$读无法预取，即 $RNP=2$；2）$C[i][j]$写无法预取，即 $WNP=1$；3）$B[k][j]$可读重用，与 i 无关，因此 $RI=1$。

表 4 列出了 Matmul 和 Dsyr2k 内核的局部性特征分析情况。按照以上的规则可以对随机符合三层嵌套-二维数据的循环进行特征抽取。

表 4 循环程序内循环操作的局部性特征

Table 4 Locality features of inner-nest operation in loop program

	ON	RP	RNP	RI	WP	WNP	WI
Matmul	$C[i][j]+=A[i][k]*B[k][j]$						
$L_3(j)$	4	2	0	1	1	0	0
$L2(k)$	4	1	1	1	0	0	1
$L_1(i)$	4	0	2	1	0	1	0
Dsyr2k	$C[j][k]+=A[i][j]*B[i][k]+B[i][j]*A[i][k]$						
$L_3(k)$	6	3	0	2	1	0	0
$L_2(j)$	6	2	1	2	0	1	0
$L_1(i)$	6	0	4	1	0	0	1

3.6 特征汇总

本节从问题规模（PS）、分块大小（TS）、循环结构（Loop Structure）以及内循环操作在每层嵌套循环的局部性表现（Locality）等方面构建了特征集，汇总成表 5 所列的应用到神经网络中的特征及其取值范围，共计 29 个有效特征。

表 5 特征及其取值范围

Table 5 Features and their range of values

Featrue Type	Featrues	values
PS	PS1	1 024～2 048
	PS2	1 024～2 048
	PS3	1 024～2 048
TS	TS	8,16,…,512
Loop Structure	Loop1($l_1=0$)	1
	Loop2($l_2=0$)	0,1
	Loop2($l_2=l_1$)	0,1
	Loop3($l_3=0$)	0,1
	Loop3($l_3=l_1$)	0,1
	Loop3($l_3=l_2$)	0,1
Locality (L_i,i=1,2,3)	L_i-RP	0,1,2,3
	L_i-RNP	0,1,2,3,4
	L_i-RI	0,1,2
	L_i-WP	0,1
	L_i-WNP	0,1
	L_i-WI	0,1
Operand Numbers	ON	4,6

4 实验与评估

4.1 神经网络

本文将 TSS 问题分解为，利用 ANN 对执行时间的回归预测和基于预测结果对分块大小进行选择。实验中，各种问题规模的特征向量都包含了第一层循环的上界（矩阵 $\boldsymbol{PS}_1$ 的尺寸）、第二层循环的上界（矩阵 $\boldsymbol{PS}_3$ 的尺寸）、第三层循环的上界（矩阵 $\boldsymbol{PS}_2$ 的尺寸）、分块大小 TS，6 个结构特征、18 个循环数据特征、1 个操作数特征，共计 29 个特征。因此，ANN 的结构是由输入层（29 个神经元）、第一隐层（32 个神经元）、第二隐层（32 个神经元）、输出层（1 个神经元）构成的全连接网络。

（1）训练集/验证集/测试集。首先，按 Kernel 类别生成和收集数据，对于每个 Kernel，PS 的大小为 1 024～2 048，生成了 500 个不同的样例，每个样例对应 64 个不同的分块组合（按照 4.3 节的选取标准），共生成 32 000 行数据用于训练，其中训练集与验证集之比为 9∶1（留数据的 10%作为验证集以进行交叉验证），按同样的方式生成另外 100 个样例（6 400 行）用于测试。为了获得每个样例的实际执行时间，需要进行启发式详尽搜索。将 6 类 Kernel 的数据集汇总，训练集、验证集与测试集的比重为 172 800∶19 200∶38 400=9∶1∶2。串行和并行实验的数据单独收集，且规模一致。

（2）数据预处理。对数据集中的每列（维）对应进行 Min-Max 标准化（Min-Max Normalization），使结果落到[0,1]区间，增强模型数据的健壮性，有助于模型的训练。转换函数如下：

$$data\_Normal_i=\frac{data_i-min_i}{max_i-min_i} \tag{3}$$

其中，$data_i$ 表示第 i 列的原始数据，max_i 为第 i 列的最大值，min_i 为第 i 列的最小值，$data\_Normal_i$ 为标准化后的第 i 列数据。

（3）Huber-loss 损失函数。它是一个用于回归问题的带参数分段损失函数，优点是能增强平方误差损失函数对离群点的鲁棒性。给定一个 δ，当预测偏差小于 δ 时，它采用平方误差，当预测偏差大于 δ 时，将 $Loss$ 减小，使用线性函数。这种方法能降低奇异数据点对于 $Loss$ 计算的权重，降低了对离群点的惩罚程度，是一种常用的鲁棒的回归损失函数。在本文模型中，超参数 δ 为 2.0，定义如下：

$$L(y,f(x))=\begin{cases}\frac{1}{2}(y-f(x))^2, & \text{if } |y-f(x)|\leqslant\delta\\ \delta\cdot\left(|y-f(x)|-\frac{1}{2}\delta\right), & \text{else}\end{cases} \tag{4}$$

（4）激活函数。tanh 的输出区间为(−1,1)，整个函数是以 0 为中心的，定义如下：

$$\tanh x=\frac{\sinh x}{\cosh x}=\frac{e^x-e^{-x}}{e^x+e^{-x}} \tag{5}$$

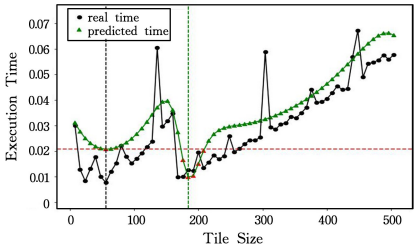
（5）优化器。采用 RMSProp 优化器，对权重 w 和偏置 b 的梯度使用微分平方加权平均数，用来修正摆动幅度，使得各个维度的摆动幅度都较小，并且也使网络函数收敛得更快。

（6）Dropout。在第二隐层和输出层之间加上 Dropout 层，参数设置为 0.8。文献[34]指出，当在较小的数据集上训练复杂的前馈神经网络时，容易造成过拟合，为了防止过拟合，可以通过阻止特征检测器的共同作用来提高神经网络的性能。Dropout 作为一种防止模型过拟合的有效技术，已被广泛应用于各种神经网络模型。

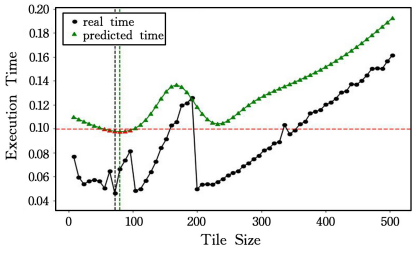
（7）其他超参数。经过经验调整，学习率取值为 0.000 5 或 0.000 1，Batch Size 选择 512。

4.2 后处理

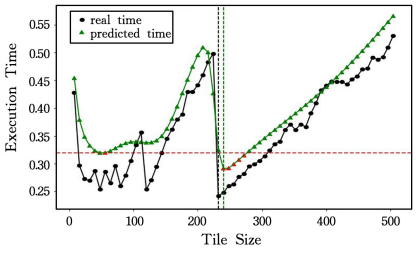
TSS 模型的输出是预测的执行时间，要得到“最佳”分块大小的预测，就需要进行后处理，根据所面临的不同问题采取不同的后处理方式。最佳分块可以是模型预测出的最短时间对应的块，我们称此模型为 TSS Top-1(1%)，简称 TSS-T1，但是由于模型存在误差，模型预测的最短时间和实际执行的真实最短时间未必一致，如果严格按照最短预测时间选择分块大小，则可能与真实最佳分块有较大的偏差。图 3 给出了在并行实验中真实执行时间与模型预测时间的拟合曲线。



(a) Trmm



(b) Matmul



(c) Dsyrk

图 3 真实执行时间与模型预测时间的对比示例(电子版为彩色)

Fig. 3 Example of comparison between real execution time and model prediction time

图 3(a)中真实的最佳分块大小为 56(黑色辅助线)，预测的最佳分块大小为 184(绿色辅助线)，两者之间的差别很大。图 3(b)、图 3(c)中真实的最佳分块分别为 72 和 232，预测的最佳分块大小分别为 80 和 240，分块大小差别不大，但是时间差别较大。为解决这个问题，从图 3(a)－图 3(c)中均可以看出，如果根据预测时间最短的前 10%(红色辅助线之下)选择分块(共 6 种候选块)，则可以捕捉到真实的最佳块。因此，对于这些候选块，我们提供了两种策略：1) TSS Top-6 (10%) Random，简称 TSS-T6-Random，从 6 个候选分块中随机选择一个，用期望时间衡量其性能；2) TSS Top-6 (10%) Search，简称 TSS-T6-Search，对 6 种分块逐个编译执行，以真实的时间为依据选出最佳块，该策略适用于迭代编译的应用场景，即用一次较高的离线编译代价换取在线部署可执行程序性能的提升。注意到，TSS-T6-Random 与 TSS-T6-Search 相比，不需要在编译阶段执行程序，因此 TSS-T6-Random 的编译效率高于 TSS-T6-Search；相反，TSS-T6-Search 通过编译阶段的尝试执行，牺牲了很小一部分时间代价换取了比 TSS-T6-Random 更好的分块性能。总之，两者各有优劣，在不同的应用场景下，应权衡性能和效率之间的利弊进行选择。

4.3 评价指标

(1)标准化时间(Normalization Time)刻画了预测的最佳分块的真实时间 T_M 和真实最佳分块的执行时间 T_O 之比，该度量刻画了预测的性能相比最佳性能的偏离程度，其越接近 1 越好。

$$T_N=\frac{T_M}{T_O} \tag{6}$$

(2)加速比(Speedup,S)的定义如下：

$$S=\frac{T_B}{T_M} \tag{7}$$

其中，T_B 为不经过分块优化的基线(Baseline)性能表现下的执行时间，本文在单核串行实验中选取 GCC(-O2)为基线，在多核并行实验中选取 GCC(-fopenmp,-O2)为基线；T_M 为采用 TSS 模型进行实验的执行时间。该指标刻画了采用的优化方法所带来的性能的加速程度。

(3)最大可用性能(Maximum Available Performance, MAP)的定义如下：

$$MAP=\frac{T_B-T_M}{T_B-T_O}\times 100\% \tag{8}$$

其中，T_B 为基准的执行时间，T_M 为所考虑的方法的执行时间，T_O($0<T_O<T$)是最佳性能的方法的执行时间；当 T_M 趋近于 T_O 时获得最大的可用性能为 100%。该度量指标越接近 100%越好。

(4)性能提升(Performance Improvement,PI)。该指标常用来比较两种方法之间的性能差异，定义如下：

$$PI=\frac{(S_a-S_b)}{S_b}\times 100\% \tag{9}$$

其中，S_a 表示 a 方法的加速比，S_b 一般表示某一基准方法的加速比。

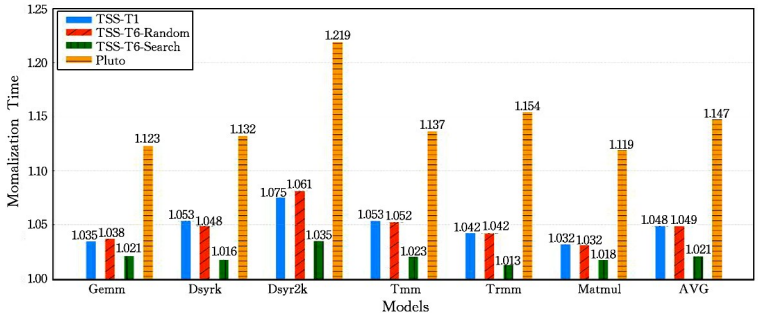
5 结果与评价

5.1 单核串行

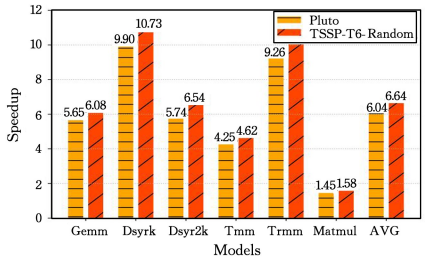
图 4(a)给出了在单核串行实验[1)]中 TSS-T1，TSS-T6，TSS-T6-Search 这 3 个模型和 Pluto 默认优化的标准化时间的对比。可以看出，3 种评估模型在每类 Kernel 上相比 Pluto 默认优化的标准化执行时间都更接近 1，即获得的分块性能更好。TSS-T1 和 TSS-T6 选择出的最佳分块性能相当，TSS-T6-Search 模型的性能稍好于前两者，当然这是用实际搜索时间换取的性能提升。对于单核串行实验，用 TSS-T6 模型已然可以获得可观的性能(标准化执行时间平均为 1.05 倍)，已逼近最佳执行时间。图 4(b)、图 4(c)分别给出了使用 TSS-T6 模型时在单核实验下针对 6 类 Kernel 测试集的加速比、最大可用性能及其平均(AVG)情况。TSS-T6 模型对于每类 Kernel 获得的加速比相比 Pluto 默认分块优化均有一定的提

1) https://github.com/CharlieCurry/TSS1.0

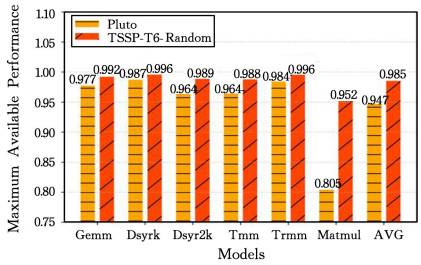
升，平均从 6.04 倍提升到 6.64 倍；最大可用性能平均从 94.7%提升至 98.5%，几乎到达了性能提升的上限。



(a)标准化执行时间



(b)加速比



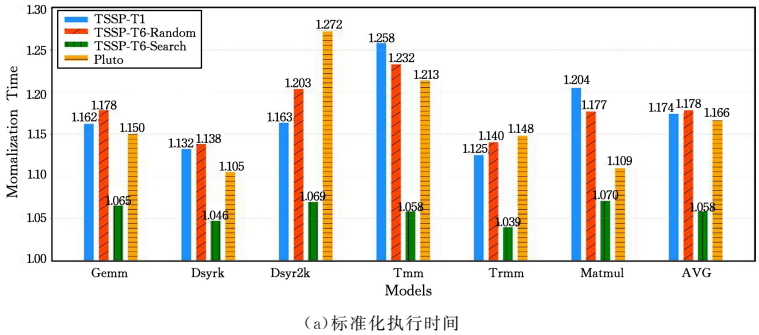
(c)最大可用性能

图 4 串行实验

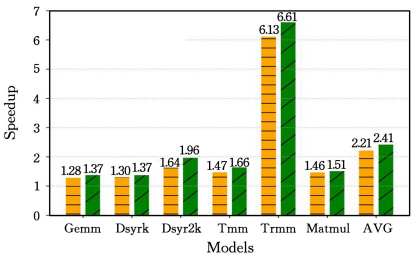
Fig. 4 Serial experiment

5.2 多核并行

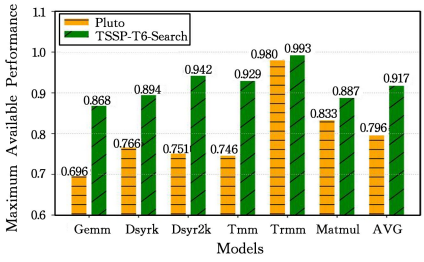
图 5(a)给出了在并行(Parallel)情形下 TSS-T1，TSS-T6-Random，TSS-T6-Search 这 3 个模型和 Pluto 默认优化的标准化执行时间的对比，为与串行模型区别开来，将这 3 种模型命名为 TSSP-T1，TSSP-T6-Random，TSSP-T6-Search。可以看出，TSSP-T1 比 TSSP-T6-Random 选择出的最佳分块性能总体要差，这是由于若 TSSP-T1 只取预测的最小执行时间则可能陷入局部最优，从而影响最佳分块的选择；TSSP-T6-Random 则是在此基础上选择预测的前 10%作为候选最佳分块，有效地避免了 TSSP-T1 陷入局部最优解，因此 TSSP-T6-Random 模型选择出的分块性能稍好；TSSP-T1 和 TSSP-T6-Random 的性能提升总体低于 Pluto，但 TSSP-T6-Search 在每个 Kernel 上均优于 Pluto，此时的平均标准化执行时间约为 1.06 倍。图 5(b)、图 5(c)分别展示了使用 TSSP-T6-Search 模型在多核并行情形下针对每类 Kernel 测试集的加速比、最大可用性能及其平均(AVG)情况。在该模型中，如图 5(b)所示，平均加速比为 2.41，优于 Pluto 的 2.21。图 5(c)则展示了 TSSP-T6-Search 模型能够达到 91.7%的最大可用性能，优于 Pluto79.6%的最大可用性能。



(a)标准化执行时间



(b)加速比



(c)最大可用性能

图 5 并行实验

Fig. 5 Parallel experiment

5.3 结果分析

多核实验由于涉及的问题更加复杂,原本在单核串行实验中表现良好的 TSSP-T6-Random 模型在多核并行实验中却表现较差,首先这是由于并行(八核)会导致每个核处理的数据规模约为原来的 1/8,这使得分块并行的收益低于分块串行的收益,从图 4(b)和图 5(b)可以看出,分块并行相对基准的收益只有分块串行相对基准收益的 1/3;TSSP-T6-Random 与 Pluto 相比,有的 Kernel 表现较好,有的 Kernel 表现较差,总体上略差,这是由本身问题的复杂性提升所导致的;其次,并行回归模型相对串行模型的误差较大,从而导致模型捕捉执行时间性能的曲线不够准确。为此,在 TSSP-T6-Random 模型的基础上从预测出的分块候选集(6 个)中进一步筛选出最佳分块(1 个),虽然对 6 个分块的实际搜索带来了时间成本的提升,但是 TSSP-T6-Search 模型的性能是值得肯定的(优于 Pluto),且时间成本在可接受的范围内。事实上,多核并行本身就是一种更大粒度意义上的分块,对本实验而言,大部分的性能收益可能出自最外层的分块方式,相反,如果再对内部循环进行分块,其收益性能的提升和带来的额外开销相比,可能就微乎其微了。总之,多核并行情况下的 TSSP-T6-Search 模型仍然能够选择出比 Pluto 性能更好的分块大小。串行实验和 Pluto 默认分块优化相比实现了平均 9.9% 的性能提升,并行实验和 Pluto 默认分块并行优化相比得到了平均 9%的性能提升。

无论是串行情况下的 TSS-T6 模型还是并行情况下的 TSSP-T6-Search 模型,都实现了找出的分块性能逼近穷尽搜索下的最佳性能,相比传统的迭代编译或穷尽搜索,在模型的部署阶段大大降低了成本,无须实际编译和运行或者只需要很少量的编译和运行。在收集数据进行穷尽搜索时耗费的时间较长(串行实验和并行实验各耗费 20 余天和 10 余天来收集相关数据),当然这个时间成本是所有模型都必须面对的;一旦将数据收集完毕,就可以在十几分钟内利用搭建好的神经网络来训练模型;进一步地,一旦 TSS 模型训练完成,就能够在 0.013 s 至 0.016 s 的时间内对特定问题规模的程序最佳分块大小进行预测。值得一提的是这种预测还可以批量进行,预测所耗费的时间并不会随预测数量的增多而线性增长;相反,如果使用穷尽搜索或迭代编译,每个用例的实际编译执行时间在数十秒到上百秒不等。总之,TSS 模型相比穷尽搜索或迭代编译有更好的效率、更低的成本,同时逼近穷尽搜索能达到的最佳性能表现。

结束语 本文探索了使用基于人工神经网络的针对特定规模的类矩阵乘法执行时间预测模型,进一步构建了最佳分块大小预测模型。首先将提取的程序原始特征作为输入,执行时间作为输出,构建基于人工神经网络的回归模型;然后根据预测的执行时间反推分块大小。本文提出了一组新的基于矩阵问题规模、分块大小、循环结构以及每层循环对内循环操作的数据预取的程序特征,以此在特定硬件条件下构建 TSS 模型。实验表明,该模型的效率优于迭代编译和穷尽搜索,性能上逼近穷尽搜索的最佳性能。未来,我们将扩充内核数量和多样性、增大问题规模、补充硬件特征、增强模型的可移植性,探索通过程序理解[35-36]技术自动提取和表示特征的方案。

参 考 文 献

[1] KASPERSKY K. Code Optimization: Effective Memory Usage [M]. A-List Publishing, 2003.

[2] ASHOURI A H, PALERMO G, CAVAZOS J, et al. Automatic Tuning of Compilers Using Machine Learning[M]. Springer International Publishing, 2018.

[3] WANG Z, O'BOYLE M. Machine learning in compiler optimization[J]. Proceedings of the IEEE, 2018, 106(11): 1879-1901.

[4] ASHOURI A H, KILLIAN W, CAVAZOS J, et al. A Survey on Compiler Autotuning using Machine Learning[J]. ACM Computing Surveys, 2018, 51(5): 1-42.

[5] LIU S, WU W G, ZHAO B, et al. Loop tiling technology for local and parallel optimization[J]. Computer research and development, 2015, 52(5): 1160-1176.

[6] YUKI T, RENGANARAYANAN L, RAJOPADHYE S, et al. Automatic creation of tile size selection models[C]// Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization. ACM, 2010: 190-199.

[7] KISUKI T, KNIJNENBURG P M W, O'BOYLE M F P. Combined Selection of Tile Sizes and Unroll Factors Using Iterative Compilation[C]// Proceedings 2000 International Conference on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2000: 237-246.

[8] STEPHENSON M, AMARASINGHE S P. Predicting unroll factors using supervised classification[C]// International Symposium on Code Generation & Optimization. IEEE, 2005.

[9] CUMMINS C, PETOUMENOS P, WANG Z, et al. End-to-end deep learning of optimization heuristics[C]// 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2017: 219-232.

[10] ASHOURI A H, MARIANI G, PALERMO G, et al. Cobayn: Compiler autotuning framework using bayesian networks[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(2): 1-25.

[11] WANG Z, O'BOYLE M F P. Mapping parallelism to multicores: a machine learning based approach[C]// ACM Sigplan Notices. ACM, 2009: 75-84.

[12] CASTRO M, GOES L F W, RIBEIRO C P, et al. A machine learning-based approach for thread mapping on transactional memory applications[C]// 2011 18th International Conference on High Performance Computing. IEEE, 2011: 1-10.

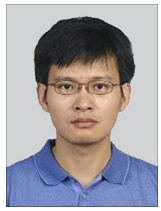
[13] KIM D G, RENGANARAYANAN L, ROSTRON D, et al. Multi-level tiling: M for the price of one[C]// Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC'07). IEEE, 2007: 1-12.

[14] MEHTA S, BEERAKA G, YEW P C. Tile size selection revisited[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2013, 10(4): 1-27.

[15] MEHTA S, GARG R, TRIVEDI N, et al. Turbotiling: Leveraging prefetching to boost performance of tiled codes[C]// Proceedings of the 2016 International Conference on Supercompu-

ting. ACM, 2016: 1-12.
[16] CHAME J, MOON S. A tile selection algorithm for data locality and cache interference[C]// Proceedings of the 13th International Conference on Supercomputing. ACM, 1999: 492-499.
[17] COLEMAN S, MCKINLEY K S. Tile size selection using cache organization and data layout[J]. ACM SIGPLAN Notices, 1995, 30(6): 279-290.
[18] YOTOV K, PINGALI K, STODGHILL P. Think globally, search locally[C]// Proceedings of the 19th annual international conference on Supercomputing. ACM, 2005: 141-150.
[19] WHALEY R C, DONGARRA J J. Automatically tuned linear algebra software [C] // Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (SC'98). IEEE, 1998: 38-38.
[20] WHALEY R C, PETITET A, DONGARRA J J. Automated empirical optimizations of software and the ATLAS project[J]. Parallel computing, 2001, 27(1/2): 3-35.
[21] SHIRAKO J, SHARMA K, FAUZIA N, et al. Analytical bounds for optimal tile size selection[C]// International Conference on Compiler Construction. Springer, Berlin, Heidelberg, 2012: 101-121.
[22] FRAGUELA B B, CARMUEJA M G, ANDRADE D. Optimal Tile Size Selection Guided by Analytical Models[C]// Parallel Computing, 2005: 565-572.
[23] LIU S, CUI Y, JIANG Q, et al. An efficient tile size selection model based on machine learning[J]. Journal of Parallel and Distributed Computing, 2018, 121: 27-41.
[24] RAHMAN M, POUCHET L N, SADAYAPPAN P. Neural network assisted tile size selection[C]// International Workshop on Automatic Performance Tuning. Berkeley, CA: Springer Verlag. 2010.
[25] MALIK A M. Optimal tile size selection problem using machine learning[C]// 2012 11th International Conference on Machine Learning and Applications. IEEE, 2012, 2: 275-280.
[26] BENGIO Y, COURVILLE A, GOODFELLOW I J. Deep learning: adaptive computation and machine learning[J]. Bengio. A. Courville, 2016, 3: 56-69.
[27] BONDHUGULA U, HARTONO A, RAMANUJAM J, et al. A practical automatic polyhedral parallelizer and locality optimizer [C]// Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2008: 101-113.
[28] ALAN J S. Cache Memories [J]. ACM Computing Surveys, 1982, 14(3): 473-530.
[29] LEVINTHAL D. Performance analysis guide for intel core i7 processor and intel xeon 5500 processors[J]. Intel Performance Analysis Guide, 2009, 30: 18.
[30] PATTERSON D A, HENNESSY J L, GOLDBERG D. Computer architecture: a quantitative approach[M]// Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers Inc. 2008, 153-162.
[31] POUCHET L N. Polybench: The polyhedral benchmark suite [J/OL]. http://www. cs. ucla. edu/pouchet/software/polybench, 2012.
[32] IPEK E, MCKEE S A, SINGH K, et al. Efficient architectural design space exploration via predictive modeling [J]. ACM Transactions on Architecture and Code Optimization (TACO), 2008, 4(4): 1-34.
[33] LEATHER H, BONILLA E, O'BOYLE M. Automatic feature generation for machine learning based optimizing compilation [C]// Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization. IEEE Computer Society, 2009: 81-91.
[34] HINTON G E, SRIVASTAVA N, KRIZHEVSKY A, et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. arXiv: 1207. 0580, 2012.
[35] LIU F, LI G, HU X, et al. Research progress of program understanding based on deep learning[J]. Computer Research and Development, 2019, 56(8): 1605-1620.
[36] JIN Z, LIU F, LI G. Program understanding: present and future [J]. Journal of Software, 2019, 30(1): 110-126.

CHI Hao-yu, born in 1996, postgraduate. His main research interests include compiler optimization and machine learning.



CHEN Chang-bo, born in 1981, Ph. D, associate professor. His main research interests include symbolic-numeric computation, automatic parallelization and optimzation of computer programs and machine learning.