**面向声子计算软件ShengBTE的GPU性能优化技术设计与实现**

摘 要

ShengBTE是计算玻耳兹曼声子输运方程常用的软件之一。虽然该软件通过计算声子散射率的收敛集从而获得晶格热导率及相关声学研究指标，但是其计算复杂度较高往往需要长时间运行才能获得计算结果。本文主要研究了声子计算软件ShengBTE在GPU上的性能优化方法。本文深入分析了ShengBTE的性能瓶颈，并针对相应的性能瓶颈提出了循环依赖消除、GPU加速和线程块调优等技术。实验结果表明，该方法在不降低精度的前提下，相对于CPU多核并行实现，在单温度实验下最高可以获得4.62倍的加速比，而在连续温度实验下最高可以获得3.97倍的加速比。

关键词：玻耳兹曼声子输运方程，ShengBTE, 性能分析优化，GPU

**Towards GPU Acceleration of Phonon Computation with ShengBTE**

**Abstract**

ShengBTE is one of the software commonly used to calculate boltzmann phonon transport equations.Although the software obtains lattice thermal conductivity and related acoustic research indexes by calculating the convergence set of phonon scattering rate, its computational complexity is high and it often takes a long time to obtain the computational results.This paper mainly studies the performance optimization method of ShengBTE on GPU.This article explores ShengBTE's performance bottlenecks and proposes techniques such as cyclic dependency elimination, GPU acceleration, and thread block tuning.Experimental results show that this method can achieve a maximum acceleration ratio of 4.62 times in the single temperature experiment and 3.97 times in the continuous temperature experiment without reducing the accuracy.

**Key words:** Boltzmann transport equation, ShengBTE, Performance analysis and optimization，GPU

# Introduction

对于热电[2]、热管理[3]和基于相变材料的非易失性存储器的开发[4]等需要具有特定导热系数材料的重要技术来讲，晶格热导率是一个重要的参数。晶格热导率是声子对总热容的贡献，而研究固体声子输运的一个重要方法是玻耳兹曼输运方程[5]。

求解晶格导热系数的研究很多。Callaway建立了一种用于计算晶格低温导热系数的模型[16]，并且于2013年由Allen改进[17]。2003年，Deinzer等人利用密度泛函扰动理论(DFPT)对三阶原子力常数(IFCs)进行了开创性的第一性原理计算，以研究Si和通用电气[18]。2005年，Yang将偏微分方程推广到多维纳米尺度热传导的研究中，包括不同的边界条件和纳米尺度热源项，研究了多维结构中瞬态声子玻耳兹曼方程(BTE)的并行求解策略，并且将the transient ballistic-diffusive heat conduction equations(BDE)的计算结果与瞬态声子BTE和傅里叶热传导方程的计算结果进行了比较[20]。2009年，Tang等人用从头开始的IFCs研究了弛豫时间近似(RTA)中MgO的导热性[19]。

从头算导热系数的成功应用的软件也很多。2014年，Tadano提出了一种计算晶体非谐力常数的系统方法。该方法采用直接法，从第一原理分子动力学模拟的高温轨迹中提取非谐力常数，应用于软件包ALAMODE[21]。2015年，Togo采用单模弛豫-时间近似方法，从一阶非谐晶格动力学计算出发，对含33种元素组合的锌闪锌矿型和纤锌矿型化合物的晶格导热系数进行了计算，得到了线性化声子玻耳兹曼方程的完整解，应用于软件包phono3py[22]。同年，Chernatynskiy引入了声子传输模拟器(PhonTS)，并支持使用多进程并行加速晶格导热系数预测[23]。

ShengBTE[]是Wu等人提出的一个新的声子玻耳兹曼输运方程的求解软件包。ShengBTE利用系统的对称性使计算更加有效，可以计算声子散射率的收敛集，并利用它们来获得晶格热导率和许多相关的数量，能够处理各向同性和各向异性晶体。此外，它还实现了作者中的一些人开发的一种近似，以有效和准确地预测纳米线的导热系数[1]。ShengBTE对于寻找具有目标导热性能的新型材料，以及深入理解固体中的热传导实验测量具有重要的价值[1]。ShengBTE是一个FORTRAN编写的纯CPU软件，支持多进程并行，但在我们的测试中发现在ShengBTE在运行时时间较长，连续温度测试时运行时间随温度数量线性增长。。通过全面的性能分析，我们发现了几个性能瓶颈，并且提出了一部分代码优化与GPU加速方法，显著提高了ShengBTE计算速度。

具体而言，本文的贡献如下：

-分析了ShengBTE的性能，确定了Ind\_plus与Ind\_minus是计算瓶颈。

-提出了几种提高性能的优化策略，包括并行化，GPU移植与线程块调优。

-在应用该算法后，我们对ShengBTE的性能进行了评估，程序在不降低迭代次数与计算精度的前提下，实现了最高4.62的加速比。

本文的其余部分组织如下。在第2节中，我们阐述了GPU加速在科学软件的应用。在第3节中，我们提供了ShengBTE的瓶颈分析。第4节给出了热点函数的优化策略。第5节阐述了实验环境，并对实验结果进行了分析。第6节介绍了相关工作，第7节给出了本文的结论。

# Performance Opportunity with GPU Acceleration

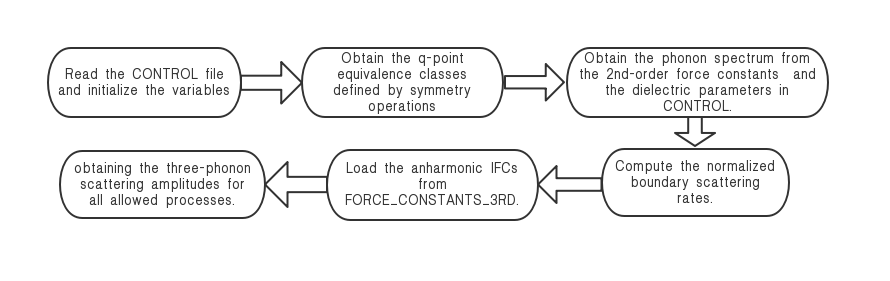
科学应用由于其计算量大、运行时间长的特点，迫切需要使用高性能的计算硬件进行加速。GPU以其大规模的并行计算单元，以及高效率的并行编程模型，例如CUDA，已经被成功应用到不同类型应用的性能加速中。特别在追求浮点运算性能的科学应用中，GPU已经得到广泛的认同并获得了巨大的成功。事实上，大部分的科学学应用已经加入了GPU并行计算模块。例如，量子化学领域的Abinit[9]，将其工作流中计算量较大的LOBPCG算法利用GPU进行并行优化，获得了3~5倍的加速比[10]；生命科学领域的冷冻电镜软件relion[11]，将E-step步骤使用GPU加速后发布了relion-2，能够动态地分配和减少GPU上的内存需求，因此甚至可以运行在个人工作站上[12]；物理学领域的AWP-ODC[34]，将三维有限差分计算移植到GPU，并且进行了线程块调优和内存优化，获得了十几倍的加速比[35]；大气环境学的区域数值天气预报模型Gales[36]，基于DALES[37]完全重新设计和重写代码，并几乎将所有计算转移到GPU；人工智能领域的Tensorflow[38]、caffe[39]等都加入了利用GPU的图像处理。NVIDIA官网给出了所有拥有GPU加速的应用列表[13]，这些应用或多或少都进行了计算密集算法的GPU移植，并且得到了数倍的加速比。图【】展示了2014年至今的使用GPU的科学应用数量变化，这一数量还在不断增加。虽然已有大量科学应用使用GPU进行性能优化，但我们注意到，声学计算软件ShengBTE还没有相关的研究工作充分发掘GPU的并行计算能力。因此，本文针对ShengBTE，分析和识别其中的计算瓶颈，并提出相应的GPU并行化方案，从而实现对ShengBTE计算性能的显著提升。

# Bottleneck Analysis

## 3.1 Execution Flow Analysis

ShengBTE运行需要三个输入文件，参数文件（CONTROL），二阶力常数文件，以及三阶原子间力常数矩阵（FORCE\_CONSTANTS\_3RD）。ShengBTE的运行流程如图【】所示。程序首先读取CONTROL文件获取参数，初始化变量，然后初步处理CONTROL数据得到对称运算定义的q点等价类。之后利用二阶力常数文件和CONTROL中的介电参数计算获取声子谱。再使用得到的声子谱计算得到谐波积分即晶格比热和小颗粒极限降低导热系数，并进行边界散射率计算和对状态的总密度和预测密度局部自适应估计。随后ShengBTE从FORCE\_CONSTANTS\_3RD加载非谐波IFCs，并最终计算所有要求的三声子散射振幅过程的量。

在所有流程中，计算三声子散射振幅过程是程序计算最昂贵的部分，CONTROL包含的一些选项参数会影响此部分的计算流程。如果convergence参数为真，则使用迭代BTE求解器(函数Ind\_\*)求解，直到实现收敛；如果为假，则在弛豫时间近似(函数RTA\_\*)中计算热导率。因此，我们将函数Ind\_\*和函数RTA\_\*使用GPU进行加速。由于函数Ind\_\*为默认执行选项，且函数Ind\_\*和函数RTA\_\*的加速方法相似，故下文中我们将不对函数RTA\_\*的加速方法进行赘述。



3.2 Bottleneck Identification

为了识别ShengBTE执行过程中的计算瓶颈，我们在5.1节所示的服务器上运行Sn2Bi-F算例，并且使用Intel的性能分析工具Vtune[14]分析了计算温度为300~900度且步长为100条件下函数运行时间。通过观察发现，ShengBTE运行中大部分时间都花费在Ind\_plus和Ind\_minus两个函数，还有少部分的MPI同步与等待时间以及极少量的其他函数运行时间。在其它算例上的ShengBTE热点函数分布与Sn2Bi-F算例基本一致。

以Sn2Bi-F为例，我们给出瓶颈分析图。如图【a】所示，300k条件下，Ind\_\*函数占据了75%的执行时间，多进程执行时产生的同步等待以及规约时间占据了23%，其余的仅占2%。在连温条件下，热点函数如图【b】，相对于单温度测试条件热点函数比例没有太大变化。因此我们重点优化Ind\_\*函数，并使用GPU加速该函数以加速ShengBTE计算过程。在ShengBTE中，程序通过Ind\_driver循环调用Ind\_plus和Ind\_minus函数，为计算密集区域。此外，MPI中有一部分产生在图【】所示的1-5步中，另一部分产生在Ind\_driver函数多进程的同步中，说明ShengBTE存在负载不均衡问题。。。

# GPU Acceleration Strategies

## 基于第3节的瓶颈分析结果，我们提出了几种优化策略，包括并行化，GPU移植以及线程块的调整来缓解计算瓶颈。

## 4.1 Eliminating the Loop-carried Dependency

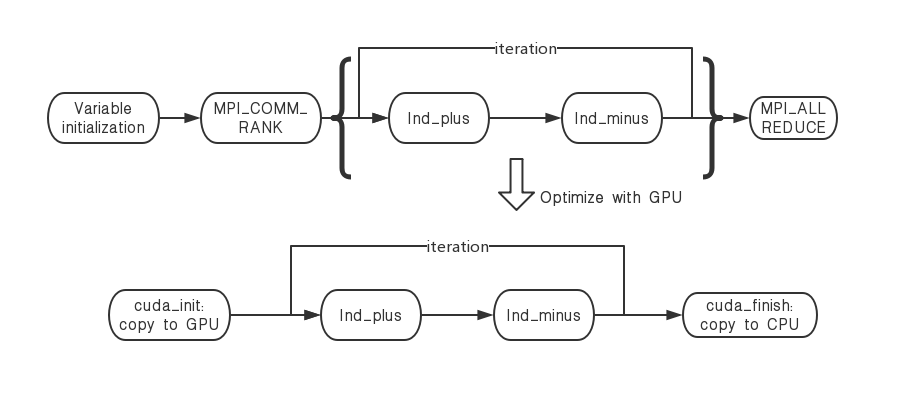
热点函数Ind\_plus和Ind\_minus有相似的结构，我们以Ind\_plus为例来介绍算法的并行化。如算法【】所示，在热点函数Ind\_plus里计算密集的部分是一个三重循环，但是在每次循环里的变量N\_plus\_count更新存在于if语句内，故我们无法提前预知每个循环内的N\_plus\_count的值。而算法第10-14行的Indof2ndPhonon\_plus，Indof3rdPhonon\_plus，Gamma\_plus数组对N\_plus\_count变量存在依赖关系，故我们无法简单的将程序并行化。为了消除循环间的依赖，我们假设N\_plus\_count在每次循环都会加一，直接取值k+Nbands\*(ii+j\*nptk)，确定了每次循环内的值。同时，扩大Indof2ndPhonon\_plus，Indof3rdPhonon\_plus，Gamma\_plus数组空间为Nbands\*nptk\*Nbands以满足计算需求。我们在循环开始前初始化Indof2ndPhonon\_plus数组为零值，在算法第10行，在满足计算条件时，Indof2ndPhonon\_plus[N\_plus\_count]得到的值(ii-1)\*Nbands+j是一个非零值，并在循环结束后以非零值为依据将三个稀疏数组稠密化。

|  |  |
| --- | --- |
| 算法 A1 Ind\_plus | |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19: | *do j=1,Nbands*  *do ii=1,nptk*  *...*  *do k=1,Nbands*  *...*  *if ((omegap.ne.0).and.(omegadp.ne.0)) then*  *...*  *if(abs(omega+omegap-omegadp).le.(2.d0\*sigma)) then*  *N\_plus\_count=N\_plus\_count+1*  *Indof2ndPhonon\_plus(N\_plus\_count)=**(ii-1)\*Nbands+j*  *Indof3rdPhonon\_plus(N\_plus\_count)=(ss-1)\*Nbands+k*  *...*  *Gamma\_plus(N\_plus\_count)=hbarp\*pi/4.d0\*WP3\*abs(Vp)\*\*2*  *Gamma\_plus(N\_plus\_count)=Gamma\_plus(N\_plus\_count)\*5.60626442\*1.d8/nptk*  *end if*  *end if*  *end do*  *end do*  *end do* |

该方法会由于Indof2ndPhonon\_plus，Indof3rdPhonon\_plus，Gamma\_plus三个数组空间增大引起调用时传输的数据量增加，并在使用GPU时显存占用会有少量增加。但是经过我们测试发现ShengBTE中数据在CPU-GPU之间传输没有成为性能限制的瓶颈，采用增加数据传输达到整体更快的加速效果是值得的。

## 4.2 Accelerating the Kenerls

。



经过4.1小节消除依赖以后，我们得到了Ind\_plus与Ind\_minus函数的无循环依赖的算法。图【】是优化前后函数执行流程，其中计算密集的区域是Ind\_plus和Ind\_minus的迭代计算。由于两个函数所需数据大量重合，我们将GPU初始化与空间释放放到了迭代的开始与结束来避免冗余的GPU数据拷贝。此外，如等式【】所示，我们在初始化时使用均分的方式在具有块GPU的节点上为进程（）绑定其对应的GPU（）来尽量保证每个GPU上的进程数相同，从而保证GPU上的负载均衡。

Ind\_plus和Ind\_minus函数则使用GPU进行加速。以Ind\_plus为例，我们把4.1小节并行化后的循环代码放入kernel调用，并在调用后把计算结果使用从GPU拷贝回CPU。需要拷贝回CPU的数据有变量WP3\_plus和4.1小节提到的三个数组，其余变量值不变或者后续不需要使用的则不用拷贝，减少数据拷贝也有利于运行速度的提升。

热点函数数据传输量较大，每个进程要占用几百兆的显存。移植完成后迭代部分运行速度显著提升，同时由于迭代部分的加速，Ind\_driver内的MPI等待时间也随之缩短。

## 4.3 Adjusting the Block Size

一个CUDA代码包含的函数主要可以分为两部分:CPU(主机)运行的函数和GPU(设备)运行的函数，在设备上运行的函数也称为内核。当CPU上的CUDA程序调用内核时，将生成一个线程网格，这个网格由块组成，每个块由线程组成。

所以在内核调用前必须指定在设备执行的函数的栅格和块的维数，栅格所有维数大小乘积等于被发送的块的数量，而块的所有维数大小乘积等于每个块的线程数量。针对4.1节所述三层循环，我们假设块大小为(block.x,block.y,block.z)，则当栅格大小为((Nbands+block.x-1)/block.x, (nptk+block.y-1)/block.y, (Nbands+block.z-1)/block.z )时我们的GPU方法可以充分使用GPU资源。由于三层循环每层大小存在巨大差异（Nbands和nptk具有数量级差距），我们发现block三个维度不会在相同时达到较优效果，而是调整到极性差距时得到了最快运行速度，这个调整相对于块大小为8x8x8的均衡状态提升了大概40%的加速效果。

# Evaluation

## 5.1 Experimental Setup

在使用的算例中，xx、xx、xx和xx来自almaBTE的数据集【】。我们使用的所有算例已经上传到【github？】。所有的实验都在本地服务器上进行，机器型号为浪潮NF5280M4，软硬件配置如表【】。由于CPU为14核，我们使用14进程并使用numactl命令绑定到单个CPU。

|  |  |
| --- | --- |
|  | Configuration |
| Machine | Inspur NF5280M4 |
| CPU | 2xIntel E5-2680v4 |
| GPU | 2xNvidia P100 |
| Memory | 384G |
| Operating System | CentOS7.6 (kernel: 3.10.0 x86 64) |
| Software | GCC v4.8.5, ICC v2018.5.274, CUDA v10.0, IFORT v2018.5.274, Intel VTune v2018.4.0.573462, Intel MPI v2018.4.274 |



## 5.2 Performance Improvement on Kernels

我们在4.2小节进行了Ind\_plus和Ind\_minus函数GPU kernel移植。Ind\_plus和Ind\_minus函数在Ind\_driver函数里循环被调用，于是我们统计了单GPU加速时整个循环的执行时间，然后与这个循环在单块CPU上的执行时间进行对比。如图【】所示，在测试得所有算例上，GPU加速效果都很明显，其中最高的是Sn2Bi-F算例，达到了5.78倍加速比，所有算例的平均加速比也有4.69倍。

## 5.3单温计算

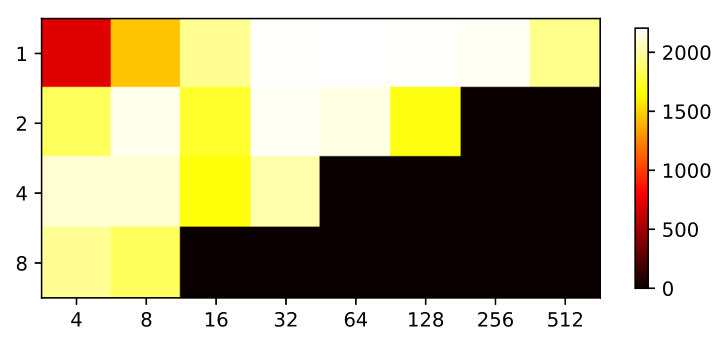
我们测试了所有优化后ShengBTE运行性能的提升。我们在CPU，CPU+GPU和CPU+2\*GPUs模式下进行了对比试验，图【时间，加速比a】展示了在温度为300条件下使用不同的七个算例计算得到的时间与加速比。单块GPU对比单块CPU最高得到了4.62倍的加速比，所有算例平均加速比为3.80倍。虽然大多数算例表现良好，但是一些算例加速比仅有1.44倍。对于加速比不高的算例penta-graphene，分析源代码与Vtune收集的数据发现其不是一个计算密集的算例。在4.1小节里我们已经给出了算法的代码结构，penta-graphene算例执行if语句内的计算次数远不如其他算例。这一点在5.2小节的kernel加速比中也可以得到验证。同时我们进行了扩展性实验，使用两块GPU运行与单块GPU运行对比。如图【时间，加速比b】，我们得到了1.4~1.8倍的加速比。最高为InYCd算例达到了1.76倍，所有算例平均加速比为1.59倍。

## 5.4 连温计算

同时，为了测试我们的优化方法在ShengBTE连续温度下的加速效果，我们使用Sn2Bi-F算例进行了以300K开始的步长为100K的多次运行测试。图【】是连温条件下在单块CPU和单块GPU上的运行时间与加速比。从图中可以看出，随着运行温度区间越来越长，GPU得到的加速比越来越高并趋于平稳。在300K温度下得到的加速比为3.97倍，逐步扩大区间后逐步增大到4.15倍，平均值也有4.10倍，此时已经趋于平缓再扩大区间也不会有太大变化。这是由于输入处理占据了一定时间，但是无论是单温度测试还是多温度测试输入处理只有一次，故温度区间越大输入处理时间占比越少，加速效果会更明显。

## 5.5 Parameter Sensitivity Analysis

在章节4.3中，我们使用GPU加速的Ind\_plus与Ind\_minus函数包含了三个参数block.x、block.y和block.z，代表GPU加速kernel中的三个维度的分块大小。为了分析这三个参数对性能的影响，我们测试了不同取值下运行xx算例的性能，并按照每个取值下的性能与最佳性能的比率创建了热力图，如图【】所示。 xxxxxx在多次测试中发现x维度和z维度保持值相等可得到更快速度。如图【】，我们在三个维度的值都选用了对所有算例都比较友好的2的幂，并进行了大量测试，以3000与运行时间t的差为数据生成了热力图。其中白色越深代表运行时间越短，红色越深代表运行时间越长，黑色部分blocksize超过限制无数据。从图中可以看出blocksize产生了两个白色聚集点，并且找到了最优化的blocksize为1x64x1。

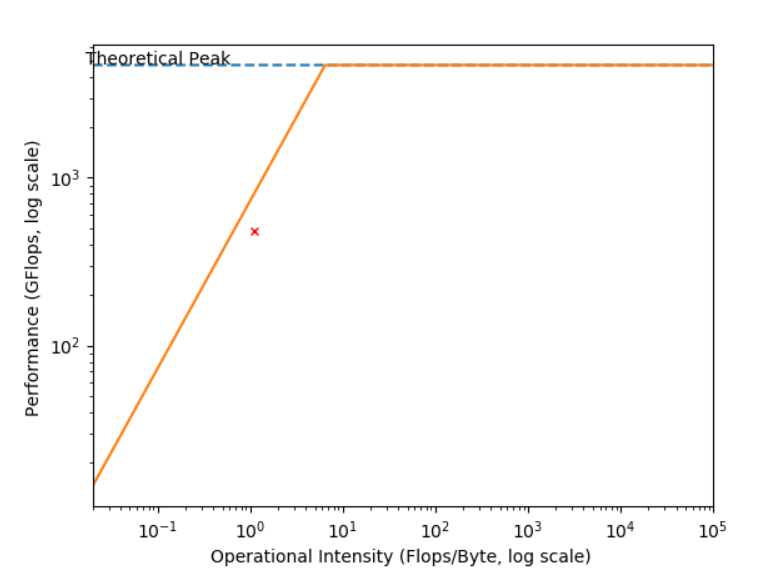
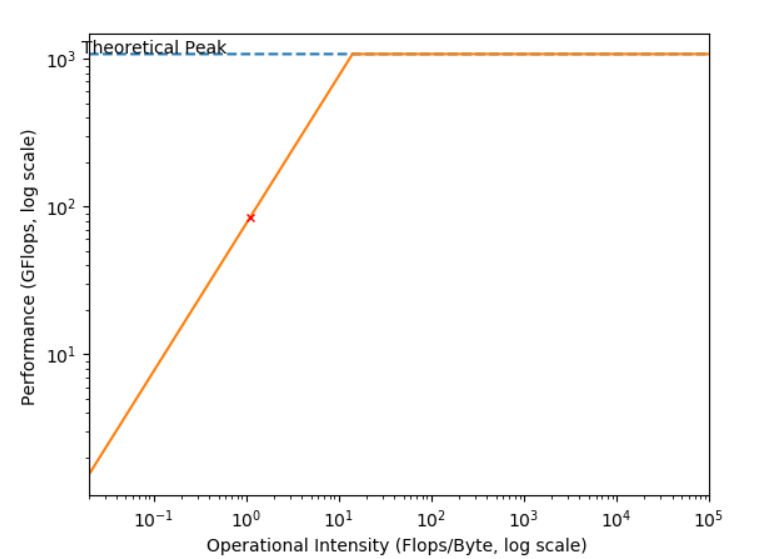


## 5.6 Roofline Model Analysis

为了深入分析硬件设施对性能的影响，我们对每个测试平台上的单节点建立了Roofline模型[31]对ShengBTE中计算核心Ind\_\*进行优化效果分析。Roofline模型可以有效将浮点性能（FLOPS）和计算密集度（Operational Intensity）之间的关系在一个二维图像中表现出来，并为优化效果、未来方向等提供参考。在Roofline模型中，屋顶（roof）代表的是处理器的峰值性能，斜坡（slope）代表的是处理器的最高内存带宽。Roofline模型中的拐点为计算受限和内存受限的分界点，计算密集度在拐点左边的为内存受限的核心，在拐点右边的为计算受限的核心。对于Nvidia P100的性能，我们在这里使用官方提供的峰值性能4.7TFlops/s[32]作为Roofline模型的屋顶，同样也使用官方提供的峰值带宽732GB/s作为其最高内存带宽。对于Intel E5-2680v4 CPU的峰值性能，我们根据官方提供的处理器参数[33]计算得出其峰值性能1.075TFlops/s，内存带宽用官方提供的峰值带宽76.8GB/s作为其最高内存带宽。

为了得到每个计算核心的计算密集度的计算公式，我们计算出了优化前后每个计算核心的浮点计算个数（FLOPS）以及数据访问量（BYTES）。在这里认为+、-、\*、/、sin、cos、sqrt、exp均为1 FLOP的双精度浮点运算。经过计算GPU实际计算速度为482.0GFlops/s，CPU实际计算速度为83.7GFlops/s，FLOPS与BYTES之比约为2.9。最终我们得到了图【】所示的roofline模型。

图【a】为CPU上的roofline model。从图中我们看出在现在的代码结构下计算性能很接近CPU性能上限，计算受CPU内存带宽限制，继续优化方向为减少数据访问量。图【b】为GPU上的roofline model，计算性能同样受内存带宽限制，但是在其他方向上还有一定优化空间。



# Related Work

2003年，Li等人实现了graphics hardware上的Lattice Boltzmann method (LBM)算法[25]，到了2009年，Kuznik等人开发了一个通用的格子玻尔兹曼代码，使之完全运行在单个GPU上[26]。2010年，Kloss等人用GPU实现求解玻尔兹曼方程的保守投影方法，并且研究了研究了二维几何求解器的优化实现方法、边界条件的设置方法、积分网格的几何实现方法和存储方法[27]。2012年，Lin等人采用多弛豫时间(MRT)和晶格玻尔兹曼方程(LBE)模拟不同腔长比(1-3腔宽深度)下的激光驱动腔流在 NVIDIA Tesla™ C2050 GPU上得到了对于 Intel Core™ i7-920 CPU20.4倍的加速比[28]。2013年，Obrecht等人将LBM算法扩展到了多GPU[29]，而到了2015年，Hong等人将其扩展到了GPU集群[30]。

almaBTE[15]是ShengBTE的继承者，两个软件的发表论文有部分相同的作者。我们下载了almaBTE的源码，并且进行了试运行。almaBTE编译产生了一组可执行文件，允许用户灵活使用。同时almaBTE保留了一个ShengBTE的模拟器，需要手动打开编译选项生成，可以处理与ShengBTE相同的输入。almaBTE产生的模拟器由FORTRAN与C混合生成，而ShengBTE完全是FORTRAN生成。我们尝试运行了多组温度下的almaBTE，发现单温度测试时alamBTE并没有比ShengBTE有运行时间上的缩短，甚至运行时间更长了。但是在连续温度下的计算时，almaBTE可以更有效地缓存声子发射/吸收过程及其相关的散射矩阵元素，这些元素与温度无关。在我们的运行中，第二个及以上的温度计算可以有3倍的加速比。遗憾的是，almaBTE没有进行GPU的代码优化。

# Conclusion and Future Work

随着材料技术发展，新的特定性能材料的寻求对现有软件计算性能的优化起到了重要的推动作用。本文对ShengBTE进行了综合性能分析，找出了性能优化的瓶颈。此外，我们还提出了一些提高ShengBTE性能的优化策略，包括并行化，GPU加速与线程块调整。实验结果表明，在不降低精度的前提下，我们的优化方法实现了单温度测试下最高4.62倍的加速比。在未来的工作中，我们希望进行knl重核CPU的移植。

# References(bib已准备好)