**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核函数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 softwares that are commonly used to calculate boltzmann phonon transport equations. Although the ShengBTE obtains lattice thermal conductivity and correlated acoustic metrics by calculating the convergence set of phonon scattering rate, its high computational complexity requires extremely long execution time to derive the simulation results. This paper mainly focuses on the performance optimization of ShengBTE on GPU. We identify the performance bottlenecks of ShengBTE and propose corresponding optimizations such as loop-carried dependency elimination, GPU kernel acceleration, and thread block tuning. The experiment results show that the proposed optimizations significantly improve the performance of ShengBTE, which achieves a maximum speedup of 4.62x in the discrete temperature simulation and 3.97x in the continuous temperature simulation without losing the simulation 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[4]等依赖于具有特定导热系数材料的重要技术来讲，确定材料的晶格热导率是一个非常重要的过程。由于热传导可以看成声子的扩散运动效果，晶格热导率即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利用玻耳兹曼输运方程[5]。

现有的研究提出了大量求解晶格导热系数的方法。Callaway建立了一种用于计算晶格低温导热系数的模型[16]，并且后续由Allen改进[17]。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]。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]。Tang等人用anharmonic IFCs研究了弛豫时间近似(RTA)中MgO的导热性[19]。

ShengBTE[1]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参数，能够处理各向同性和各向异性晶体。此外，它还可以利用近似方法，以快速、有效地预测纳米线的导热系数。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。ShengBTE是一个FORTRAN编写的纯CPU软件，支持多进程并行。然而，目前的ShengBTE存在运行时时间较长的缺点，特别是连续温度模拟时运行时间随温度数量线性增长，阻碍了ShengBTE的广泛应用。针对上述问题，本文对ShengBTE进行了深入的性能分析，识别出了导致性能瓶颈的若干热点函数，并提出了相应的性能优化与GPU加速方法，显著提高了ShengBTE计算性能。

具体而言，本文的贡献如下：

-我们深入分析了ShengBTE的性能瓶颈，识别了scattering amplitudes of absorption processes(Ind\_plus)与scattering amplitudes of emission processes(Ind\_minus)是计算过程中的主要性能热点。

-我们提出了若干提高ShengBTE性能的优化方法，包括循环依赖消除，GPU核函数优化与线程块调优。

-我们选择具有代表性的数据集对优化后的ShengBTE进行了性能评估，实现结果显示在不降低模拟精度的前提下，本文提出的优化方法实现了最高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在加速科学应用中的使用，这也激发了本文的工作。在第3节中，我们提供了ShengBTE的性能瓶颈分析。第4节给出了热点函数的优化方法。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][10]，将其工作流中计算量较大的LOBPCG算法利用GPU进行并行优化；生命科学领域的冷冻电镜软件relion[11][12]，将期望最大化算法中的计算最密集部分使用GPU加速，并能够动态地分配计算任务并适配GPU上的内存需求；物理学领域的AWP-ODC[34][35]，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；大气环境学的区域数值天气预报模型Gales[36]，基于DALES[37]重新设计程序结构，并几乎将所有计算转移到GPU；人工智能领域的Tensorflow[38]、caffe[39]等都使用GPU加速神经网络的训练、推理过程。图1展示了2014年至今的使用GPU的HPC应用数量变化[43]，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优化方案，从而实现对ShengBTE计算性能的显著提升。

Figure 1: The number of GPU-Accelerated HPC applications.

# 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2所示。程序首先读取CONTROL文件获取参数，初始化变量，然后处理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，故下文中我们将针对函数Ind\_\*的加速方法进行介绍。

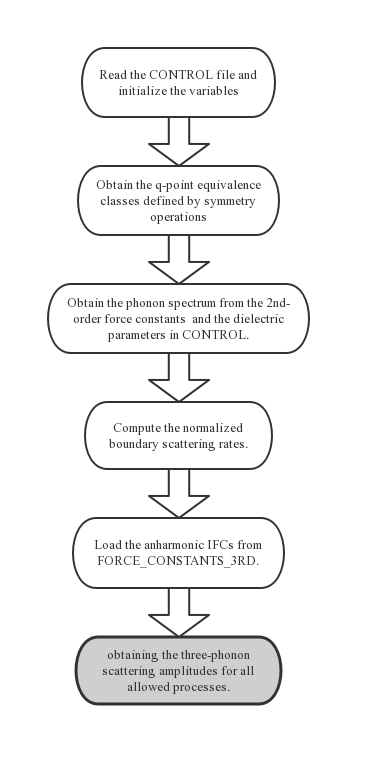


Figure 2: The execution workflow of ShengBTE

3.2 Bottleneck Identification

为了识别ShengBTE执行过程中的计算瓶颈，我们在5.1节所示的实验服务器上运行Sn2Bi-F算例，并且使用Intel的性能分析工具Vtune[14]分析了单温度300K计算（图3(a)）和连续温度300~900K计算且步长为100K（图3(b)）条件下函数运行时间。

(a) (b)

Figure 3. The execution time distribution at (a) 300K, and (b) 300K-900K.

如图3(a)所示，在300k温度条件下，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3(b)，相对于单温度测试条件热点函数比例没有太大变化。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。在其它算例上的ShengBTE热点函数分布与Sn2Bi-F算例基本一致。因此我们重点考虑使用GPU优化Ind\_\*函数以加速ShengBTE计算过程。

# GPU Acceleration Strategies

## 针对第3节的性能瓶颈分析结果，我们提出了三种性能优化方法，包括循环依赖消除，GPU核函数加速以及线程块调优来加速计算瓶颈。

## 4.1 Eliminating the Loop-carried Dependency

由于热点函数Ind\_plus和Ind\_minus有相似的结构，我们以Ind\_plus为例来介绍消除循环依赖从而实现算法的并行化。如算法1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（算法1行16）存在于if语句内，无法提前预知每个循环内的N\_plus\_count的值。而算法1第17、18、23和24行的Indof2ndPhonon\_plus，Indof3rdPhonon\_plus，Gamma\_plus数组对N\_plus\_count变量存在依赖关系，进而无法简单的将程序并行化。为了消除循环间的依赖，我们可以先假设N\_plus\_count在每次循环都会加一，因此可以将N\_plus\_count直接取值为k+Nbands\*(ii+j\*nptk)（算法2第17行），从而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，并在初始化Indof2ndPhonon\_plus数组为零值（算法2第8行）。由于在满足计算条件时，Indof2ndPhonon\_plus[N\_plus\_count]为非零值（算法2第18行），因此我们在循环结束后以非零值为依据将三个稀疏数组稠密化以得到最后结果（算法2第32-39行）。最终优化后的算法如算法2所示。

|  |  |
| --- | --- |
| **算法 1 Original algorithm of** Ind\_plus | |
| **1:**  **2:**  **3:**  **4:**  **5:**  **6:**  **7:**  **8:**  **9:**  **10:**  **11:**  **12:**  **13:**  **14:**  **15:**  **16:**  **17:**  **18:**  **19:**  **20:**  **21:**  **22:**  **23:**  **24:**  **25:**  **26:**  **27:**  **28:**  **29:** | *do j=1,Nbands*  *do ii=1,nptk*  *qprime=IJK(:,ii)*  *realqprime=matmul(rlattvec,qprime/dble(ngrid))*  *omegap=energy(ii,j)*  *fBEprime=1.d0/(exp(hbar\*omegap/Kb/T)-1.D0)*  *do k=1,Nbands*  *qdprime=q+qprime*  *qdprime=modulo(qdprime,Ngrid)*  *realqdprime=matmul(rlattvec,qdprime/dble(ngrid))*  *ss=Index\_N(qdprime(1),qdprime(2),qdprime(3))*  *omegadp=energy(ss,k)*  *if ((omegap.ne.0).and.(omegadp.ne.0)) then*  *sigma=scalebroad\*base\_sigma(velocity(ii,j,:)-velocity(ss,k,:))*  *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*  *fBEdprime=1.d0/(exp(hbar\*omegadp/Kb/T)-1.D0)*  *Vp=Vp\_plus(i,j,k,list(ll),ii,ss,realqprime,realqdprime,eigenvect,&*  *Ntri,Phi,R\_j,R\_k,Index\_i,Index\_j,Index\_k)*  *WP3=(fBEprime-fBEdprime)\*exp(-(omega+omegap-omegadp)\*\*&*  *2/(sigma\*\*2))/sigma/sqrt(Pi)/(omega\*omegap\*omegadp)*  *WP3\_plus=WP3\_plus+WP3*  *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* |

|  |  |
| --- | --- |
| **算法 2 Optimized algorithm of** Ind\_plus | |
| **1:**  **2:**  **3:**  **4:**  **5:**  **6:**  **7:**  **8:**  **9:**  **10:**  **11:**  **12:**  **13:**  **14:**  **15:**  **16:**  **17:**  **18:**  **19:**  **20:**  **21:**  **22:**  **23:**  **24:**  **25:**  **26:**  **27:**  **28:**  **29:**  **30:**  **31:**  **32:**  **33:**  **34:**  **35:**  **36:**  **37:**  **38:**  **39:** | *do j=1,Nbands*  *do ii=1,nptk*  *qprime=IJK(:,ii)*  *realqprime=matmul(rlattvec,qprime/dble(ngrid))*  *omegap=energy(ii,j)*  *fBEprime=1.d0/(exp(hbar\*omegap/Kb/T)-1.D0)*  *do k=1,Nbands*  *Indof2ndPhonon\_plus(k+Nbands\*(ii+j\*nptk))=0*  *qdprime=q+qprime*  *qdprime=modulo(qdprime,Ngrid)*  *realqdprime=matmul(rlattvec,qdprime/dble(ngrid))*  *ss=Index\_N(qdprime(1),qdprime(2),qdprime(3))*  *omegadp=energy(ss,k)*  *if ((omegap.ne.0).and.(omegadp.ne.0)) then*  *sigma=scalebroad\*base\_sigma(velocity(ii,j,:)-velocity(ss,k,:))*  *if(abs(omega+omegap-omegadp).le.(2.d0\*sigma)) then*  *N\_plus\_count= k+Nbands\*(ii+j\*nptk)*  *Indof2ndPhonon\_plus(N\_plus\_count)=(ii-1)\*Nbands+j*  *Indof3rdPhonon\_plus(N\_plus\_count)=(ss-1)\*Nbands+k*  *fBEdprime=1.d0/(exp(hbar\*omegadp/Kb/T)-1.D0)*  *Vp=Vp\_plus(i,j,k,list(ll),ii,ss,realqprime,realqdprime,eigenvect,&*  *Ntri,Phi,R\_j,R\_k,Index\_i,Index\_j,Index\_k)*  *WP3=(fBEprime-fBEdprime)\*exp(-(omega+omegap-omegadp)\*\*&*  *2/(sigma\*\*2))/sigma/sqrt(Pi)/(omega\*omegap\*omegadp)*  *WP3\_plus=WP3\_plus+WP3*  *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*  *ii=1*  *do j=1, Nbands\*nptk\*Nbands*  *if(Indof2ndPhonon\_plus(j).ne.0) then*  *Indof2ndPhonon\_plus(ii)=Indof2ndPhonon\_plus(j)*  *Indof3rdPhonon\_plus(ii)=Indof3rdPhonon\_plus(j)*  *Gamma\_plus(ii)=Gamma\_plus(j)*  *ii=ii+1*  *end if*  *end do* |

算法2可以有效地消除循环间之间的依赖，从而为程序在GPU上的并行化提供了可能。但该方法可能会由于Indof2ndPhonon\_plus，Indof3rdPhonon\_plus，Gamma\_plus三个数组空间增大而引起调用时传输的数据量增加，并造成计算过程中GPU显存占用率增加。但是经过我们测试发现，上述改进后的数组在CPU-GPU之间的传输没有成为ShengBTE的性能瓶颈，因此上述方法的加速效果有效的。

## 4.2 Accelerating the Kenerls

经过4.1小节消除循环依赖以后，我们得到了Ind\_plus与Ind\_minus函数的无循环依赖的算法（例如算法2），而这两个函数都通过Ind\_driver函数迭代调用。Ind\_driver函数首先初始化变量，然后进行多进程间任务的分配，随后在每个进程中计算迭代的声子发射吸收求解过程，最后在所有进程间同步计算结果。优化前后计算流程如图4所示。

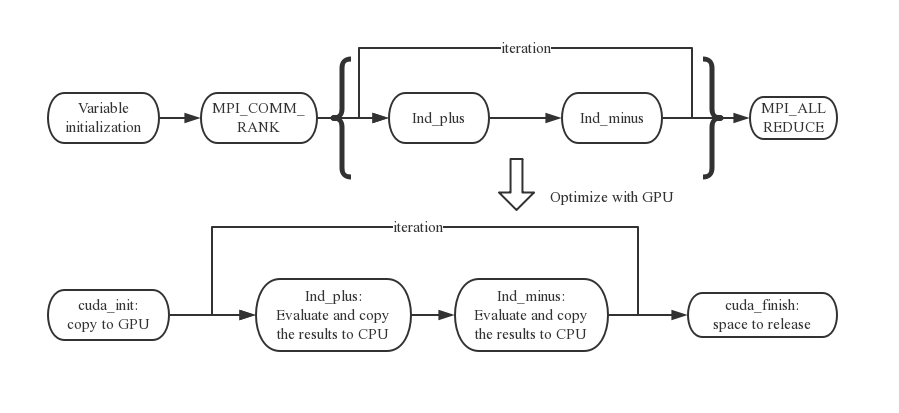


Figure 4：The execution flow of the Ind\_driver function before and after optimization.

如图4所示，计算密集的区域是Ind\_plus和Ind\_minus的迭代计算，我们使用GPU加速这两个过程。在数据传输方面，由于两个函数所需数据大量重叠，我们在迭代的开始与结束时完成所有的GPU初始化、所需数据的初始化与传输以及GPU空间的释放来避免冗余的GPU数据拷贝。此外，如公式1所示，我们在初始化时使用取模的方式在具有块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，进而缩短MPI的等待同步时间。

公式 1

这里以Ind\_plus为例，具体介绍对Ind\_plus和Ind\_minus核函数使用GPU进行加速。我们把4.1小节消除循环依赖后的三重循环部分（算法2第1、2和7行）封装为GPU kernel函数，每次循环迭代则映射为GPU上的一个线程，并在计算完成后将结果从GPU拷贝回CPU。具体来讲，我们使用三维线程块(block.x，block.y，block.z)，其中每个线程块中x维度中含有block.x线程，映射到算法2中的j循环变量；y维度中含有block.y线程，映射到算法2中ii循环变量；z维度中含有block.z线程，映射到算法2中k循环变量。循环中的计算部分（算法2行8-27）按照该线程块分配GPU线程中。由于4.1小节中我们已经消除了循环依赖，因此GPU线程可以并行执行而不需要同步操作，大大增加了程序在GPU上的并行度。此外，GPU加速后Ind\_plus函数需要拷贝回CPU的数据只有变量WP3\_plus和4.1小节提到的三个数组，其余值未发生变化或者后续不需要使用的变量则不用拷贝，进一步减少需要拷贝的数据量，提升运行速度。Ind\_minus函数的GPU加速方法与Ind\_plus函数类似。

在实现上，虽然ShengBTE为纯FORTRAN编写的应用，但我们在GPU加速上使用了FORTRAN-C-CUDA混合编程（FORTRAN调用C接口封装的CUDA-C加速代码）的形式，而没有使用PGI支持的FORTRAN-CUDA的加速实现方案[41]。我们选择该实现方案的原因有二：1) PGI支持的FORTRAN-CUDA会带来冗余的数据拷贝，在性能上不如使用FORTRAN-C-CUDA混合编程；2) 使用FORTRAN-C-CUDA可以在CPU代码（如图2中第1-5步）上可以使用Intel ICC编译器对代码进行编译优化，从而保证更佳的性能。

## 4.3 Tuning the Thread Block

由于CUDA编程模型在组织GPU线程时采用了网格（Grid）、线程块（Block）和线程（Thread）三层组织形式，首先将GPU上的计算单元分为若干个网格，每个网格内包含若干个线程块，每个线程块包含若干个线程。在GPU核函数调用前必须指定函数的网格和块组织方式，网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。不同的线程组织方式会对核函数性能产生显著影响。针对4.1节所述三层循环，我们使用三维块(block.x,block.y,block.z)组织线程。由于三层循环每层大小存在巨大差异（Nbands和nptk存在数量级差距），因此block三个维度的取值范围需要与不同的循环大小相对应。此外，我们设置网格大小为((Nbands+block.x-1)/block.x, (nptk+block.y-1)/block.y, (Nbands+block.z-1)/block.z )，这时我们提出的GPU加速方法可以充分利用GPU资源。5.5小节中详细给出了三维线程块(block.x,block.y,block.z)不同维度取值对ShengBTE加速效果的影响。

# Evaluation

## 5.1 Experimental Setup

我们使用almaBTE的数据集[40]对优化后的ShengBTE性能进行验证，该数据集包括SbCaK、CoSiTa、InYCd和RuAsV。所有的实验在如表1所示的服务器上进行。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对优化前后的ShengBTE进行测试。

Table 1： The hardware and software configurations

|  |  |
| --- | --- |
|  | Configuration |
| 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优化。由于Ind\_plus和Ind\_minus函数在Ind\_driver函数里进行了多次迭代，这里我们给出单GPU加速时整个Ind\_driver函数迭代过程的执行时间，然后与该迭代过程在单块CPU上的执行时间进行对比。如图5所示，在测试的所有算例上，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。

Figure 5：The execution time and performance speedup of the Ind\_driver function under different datasets.

## 5.3单温度计算

我们在CPU，CPU+GPU和CPU+2\*GPUs模式下对ShengBTE单温度计算的整体性能进行了实验验证，图6展示了在温度为300K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，通过分析源代码与Intel Vtune收集性能数据发现其不是一个计算密集的算例，其执行有效计算次数（算法1行16-26）远不如其他算例。这一点在5.2小节的kernel加速比（图5中graphene）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7所示。在所有算例中两块GPU可以获得1.45~1.76倍的加速比，其中最高为InYCd算例达到了1.76倍，所有算例平均加速比为1.59倍。

Figure 6：The execution time and performance speedup of our optimized ShengBTE under different datasets.

Figure 7：The scalability of our optimized ShengBTE across multiple GPUs under different datasets.

## 5.4 连续温度计算

同时，为了测试我们的GPU优化方法在ShengBTE连续温度模拟下的加速效果，我们使用Sn2Bi-F算例进行了以300K温度开始的步长为100K的多次计算。图8是连温条件下ShengBTE在单块CPU和两块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，随着模拟温度区间扩大而逐步增加到4.15倍，平均加速比可以达到4.10倍，而当模拟温度区间超过800K后，加速比趋于平缓不会再有明显提升。这是由于输入数据前处理占据了执行时间的一定比例，但是无论是单温度还是连续温度模拟中输入前处理只需要只有一次，因此温度区间越大输入前处理时间占执行时间的比重比越小，加速效果会更明显。然而，当温度区间超过一定范围后，前处理期间所占执行时间比例可以忽略，再次增加温度区间也很难得到更好的加速效果，因此加速比趋于平缓。

Figure 8：The execution time and performance speedup of our optimized ShengBTE under continuous temperature simulations with Sn2Bi-F dataset. The stride of the temperature is 100K.

## 5.5 Parameter Sensitivity Analysis

在4.3小节中，我们讨论了设置线程块参数block.x、block.y和block.z会对使用GPU加速的Ind\_plus与Ind\_minus函数性能产生影响。为了进一步定量分析，我们测试了不同参数取值组合下运行Sn2Bi-F算例的性能，并按照每个取值组合下的性能与最佳性能的比率创建了热力图，如图9所示。由于循环的第一层与第三层次数相同，栅格在x维度被充分利用时z维度取同样的值。我们以x(z)维度取值为纵轴，y维度取值为横轴，在三个维度的值都选用了对GPU性能友好的2的幂。其中颜色越浅代表运行时间越短，颜色越深代表运行时间越长，黑色部分表示blocksize超过限制没有实验数据。从图中可以看出线程块的最优设置为1x64x1。

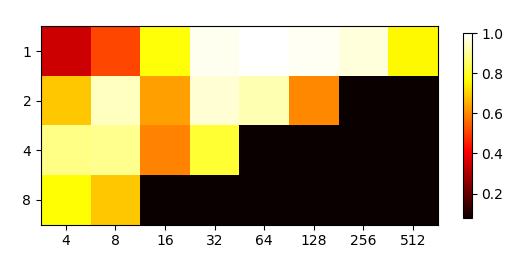


Figure 9：The sensitive analysis of blocking parameters (block.x, block.y, block.z) in iterations of Ind\_driver. Each cell is the relative performance of the GPU accelerated kernel.

## 5.6 Roofline Model Analysis

为了深入理解本文提出的GPU优化方法的效果，以及未来进一步优化的方向，我们建立了Roofline模型[31]对ShengBTE中计算核心Ind\_drive的优化效果进行分析。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。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。

为了得到函数Ind\_drive的计算密集度，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。使用Sn2Bi-F算例经过测试GPU加速的Ind\_drive中迭代过程的实际计算性能为482.0GFlops/s，原CPU版本的实际计算性能为83.7GFlops/s，GPU加速和CPU版本的Ind\_drive计算密度均可看作其计算核心Vp\_\*函数的计算密度，计算公式为(3+Ntri\*(34+27\*16))/(Ntri\*(3+27\*12))≈1.42。最终得到的Roofline模型如图10所示。

图10中的红线为CPU上的roofline model，红色叉代表原CPU版本的Ind\_drive函数的计算性能。从图10中我们看出原CPU算法的计算性能受CPU内存带宽限制，需要增加其算法的计算密集度来增加其性能上限。蓝线为GPU上的roofline model，蓝色叉代表GPU加速的Ind\_drive函数的计算性能。可以看出，GPU优化后的性能与原始的CPU版本相比获得了显著提升，但仍然离GPU理论性能上限还有一定距离。同时我们也注意到，GPU优化后的性能已经依然受限于GPU内存带宽，为了进一步提升性能，我们需要提升Ind\_drive函数在GPU上的计算密度从而逼近GPU的理论性能上限。

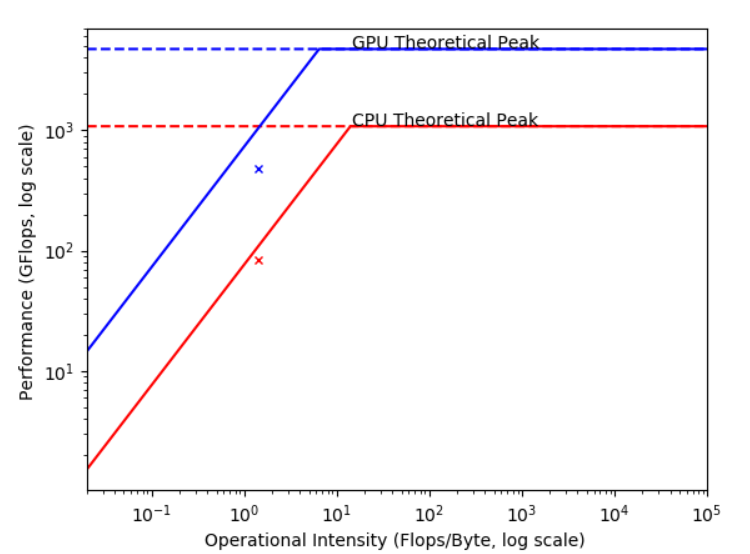


Figure 10：The roofline model the original (on CPU) and optimized (on GPU) Ind\_drive function.

# Related Work

计算导热系数并得到广泛应用的软件有很多。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]。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]。Chernatynskiy引入了声子传输模拟器(PhonTS[23])，并支持使用多进程并行加速晶格导热系数预测。ShengBTE软件包[1]基于声子玻耳兹曼输运方程，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参数，能够处理各向同性和各向异性晶体。almaBTE[15]在连续温度下可以更有效地缓存与温度无关的声子发射/吸收过程及其相关的散射矩阵元素，从而获得客观的加速效果，但其在单温度计算上的性能不如ShengBTE。然而，上述软件没有使用GPU对其计算进行加速，并且缺乏GPU上的性能优化方法。本文的工作正是针对ShengBTE软件在GPU上进行性能优化，从而降低声学计算过程的时间开销。

玻尔兹曼方程求解在GPU上的实现和优化存在大量的研究工作。Li等人[25]实现了graphics hardware上的Lattice Boltzmann method (LBM)算法。Kuznik等人[26]开发了一个通用的格子玻尔兹曼GPU加速算法，使之完全运行在单个GPU上。基于他们的研究成果，Obrecht等人[29]将LBM算法扩展到了多GPU，而Hong等人[30]将其扩展到了GPU集群。此外，Kloss等人[27]用GPU实现求解玻尔兹曼方程的保守投影方法，并且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。Lin[28]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，并在 NVIDIA GPU上得到了20.4倍的加速比。然而，目前没有相关的研究工作对ShengBTE以及其实现的玻耳兹曼声子输运方程求解算法在GPU上实现和优化。

# Conclusion and Future Work

本文对ShengBTE进行了深入的性能分析，识别出了若干阻碍其性能提升的热点函数。为此，我们提出了一系列提高ShengBTE性能的优化方法，包括循环依赖消除、GPU核函数加速和线程块调整。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模拟下最高4.62倍的加速比，连续温度模拟下最高3.97倍的加速比。在未来的工作中，我们希望对ShengBTE运行过程中的负载均衡进行优化。同时，我们希望在更多的众核处理器上例如KNL对ShengBTE进行性能优化。

# References(bib已准备好)