基于指导语句的异构协同并行编程框架的研究与实现

——阳王东
——湖南大学

报 告 内 容

一、研究背景

二、异构协同并行编程总体框架

三、异构协同计算中任务调度

四、异构协同计算中通信优化

五、原型实现及应用案例

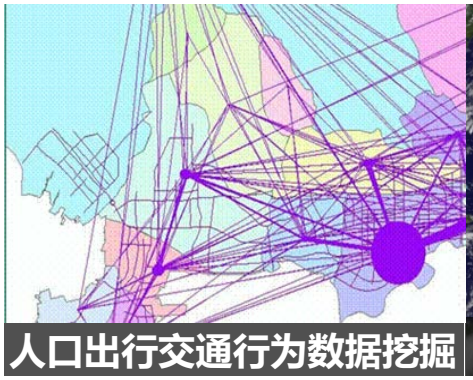
1.研究背景-异构众核系统迅速崛起

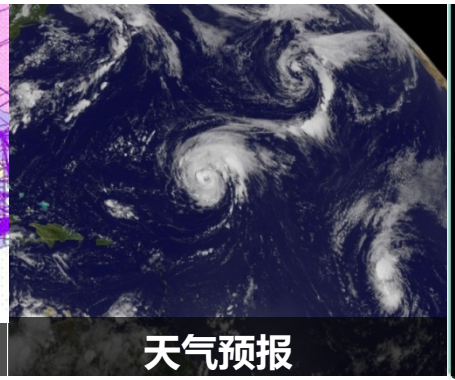
异构众核体系结构因其高性能，低能耗，低成本的优势，成为当今和未来高性能计算发展的重要趋势

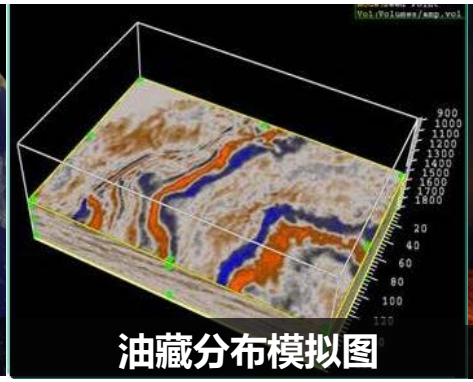
天河二号	CPU核心数：38.4万 MIC核心数：273.6万	MIC峰值比重 87.7%
Titan	CPU核心数：29.9万 GPU核心数：4664.5万	GPU峰值比重 80.6%
Piz Daint	CPU核心数：8.4万 GPU核心数：1315.9万	GPU峰值比重 87.5%

不能有效利用异构众核，将浪费70%的异构计算能力。

1.研究背景-异构计算应用日益增多


人口出行交通行为数据挖掘


天气预报


油藏分布模拟图


核爆炸模拟

异构众核系统的计算能力日益强大，**应用需求日益增多**，但访存墙、通信墙、可靠性墙和**编程墙**严重**制约基于异构平台的应用**，也难以充分发挥异构系统的性能优势。


神威蓝光

数万CPU核

天河二号

数百万异构核心

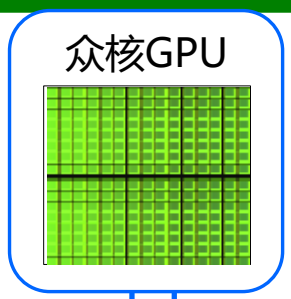

泰坦

1. 研究背景-异构计算中可编程性面临挑战

CPU并行编程模型: OpenMP

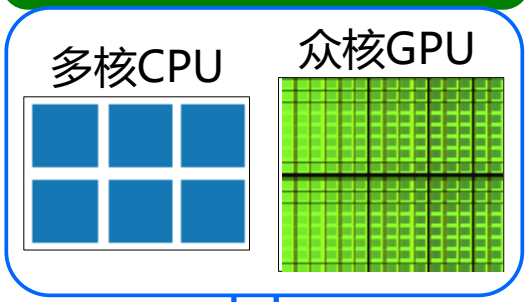
问题： 仅能发挥多核CPU的计算潜能

GPU并行编程模型: CUDA



问题： 仅能发挥众核加速器的计算潜能

混合并行编程模型: OpenMP+CUDA



问题： 手动管理各处理器之间的任务划分、负载均衡、通信优化等，复杂而繁琐

挑 战: 如何减轻基于异构平台的编程负担并有效利用异构众核？设计一种**简单易用**的**异构协同并行编程框架**，能够**自动**地在各处理器之间**划分任务**并管理各处理器之间的**负载均衡**和**通信优化**等。

报 告 内 容

一、研究背景

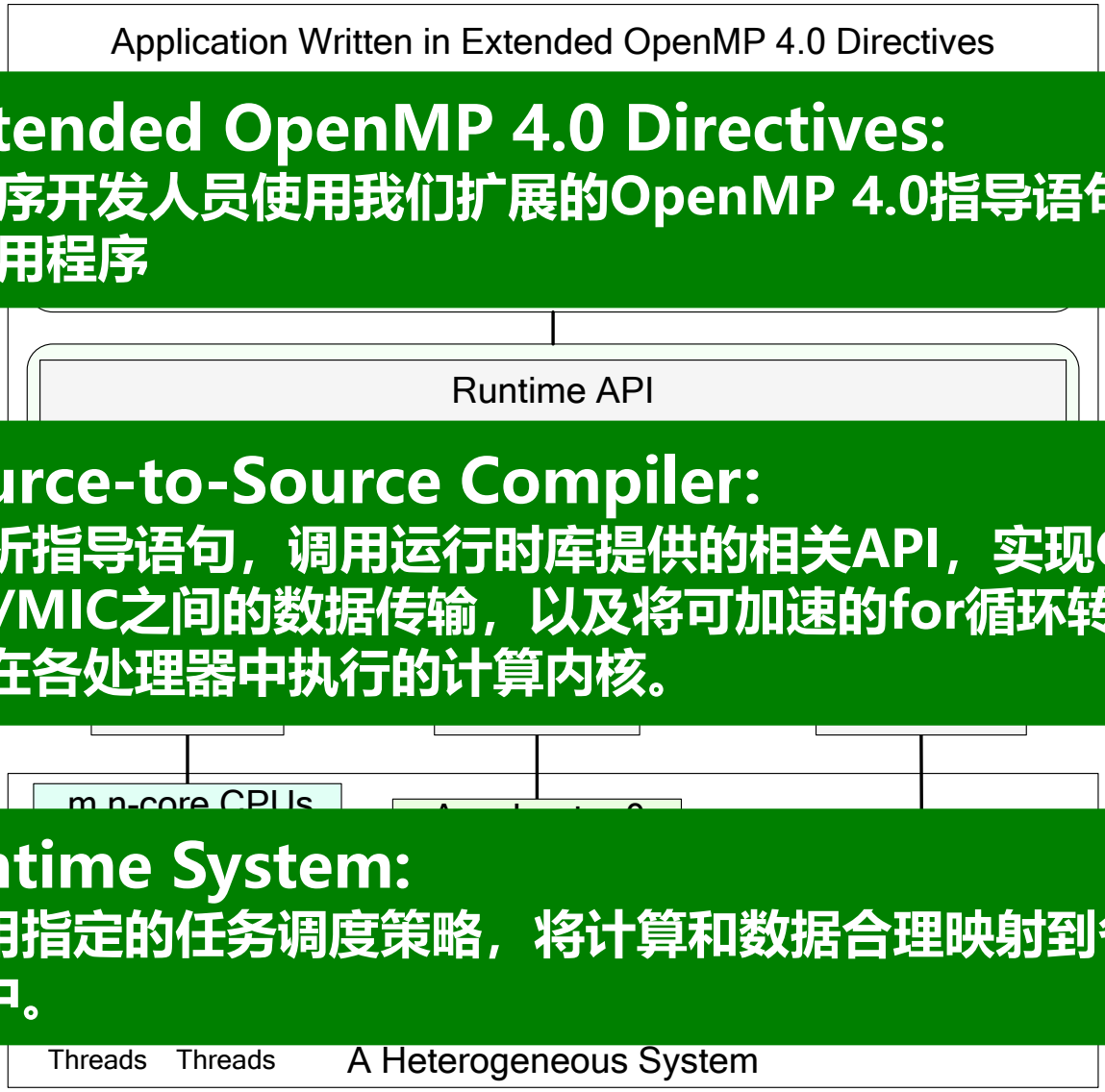
二、异构协同并行编程总体框架

三、异构协同计算中任务调度

四、异构协同计算中通信优化

五、原型实现及应用案例

2. 异构协同并行编程总体框架



1. Extended OpenMP 4.0 Directives:
应用程序开发人员使用我们扩展的OpenMP 4.0指导语句来编写应用程序

2. Source-to-Source Compiler:
负责解析指导语句，调用运行时库提供的相关API，实现CPU与GPU/MIC之间的数据传输，以及将可加速的for循环转换成能够在各处理器中执行的计算内核。

3. Runtime System:
负责采用指定的任务调度策略，将计算和数据合理映射到各处理器中。

OpenHCPP: 一个适应于异构系统的基于指导语句的协同并行编程框架

2. 异构协同并行编程总体框架

OpenHCPP: 一个适应于异构系统的基于指导语句的协同并行编程框架

```
1 int M = 1024, N = 1024, L = 1024;
2 //allocate host memory for A[M*L], B[L*N] and C[M*N]
3 //randomly initialize A and B
4 #pragma omp target teams distribute parallel for devices(cpu, gpu:0) \
5 map(to: M, N, L, B[0:L*N]) map(part to: A[0:M*L:L]) \
6 map(part from: C[0:M*N:N]) \
7 num_teams(gpu:0:128) num_threads(gpu:0:128) \
8 distribution(dynamic_preemption(32))
9 for(int i = 0; i < M; i++) {
10   for(int j = 0; j < N; j++) {
11       double sum = 0;
12       for(int k = 0; k < L; k++)
13           sum += A[i*L+k] * B[k*N+j];
14       C[i*N+j] = sum;
15   }
16 }
17 //deallocate host memory for A, B and C
```

An Example: Matrix Multiplication

报 告 内 容

一、研究背景

二、异构协同并行编程总体框架

三、异构协同计算中任务调度

四、异构协同计算中通信优化

五、原型实现及应用案例

3. 异构协同计算中任务调度

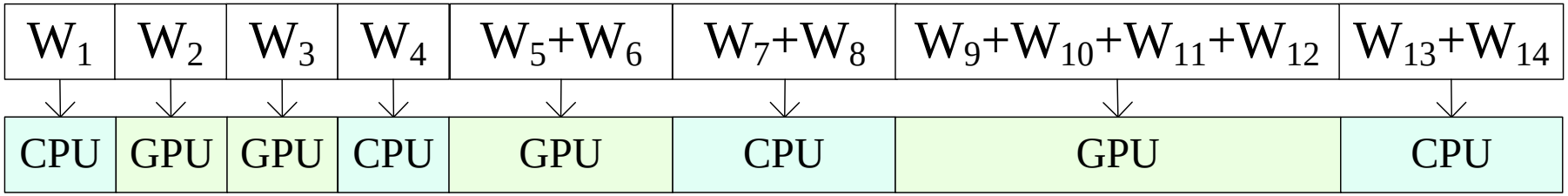
任务调度策略 1: Feedback-based Dynamic Task Scheduling

在异构协同编程中，**手动管理**各处理器之间的**任务划分**和**负载均衡**，复杂而繁琐，为此，我们的编程框架提供了两种任务调度策略，**自动管理**各处理器之间的**任务划分**，并保证**各处理器**都能够得到**充分利用**。

FDTS: 循环地从任务队列中获取任务，根据当前的任务划分比例将取出的任务合理分配给各处理器，待各处理器协同执行完毕之后，将实际测量得到的执行时间和实际工作负载相结合，更新任务划分比例，作为下次任务划分的依据。

3. 异构协同计算中任务调度

任务调度策略 2: Preemption-based Dynamic Task Scheduling



基于抢占式的动态任务调度

$$If\ \ Vgpu.curr\ >\ Vgpu.prev\times(1+\alpha)\ ,\ \ then\ Wgpu.next\ =\ Wgpu.curr\times 2;$$
$$If\ \ Vgpu.curr\ <\ Vgpu.prev\times(1-\alpha)\ ,\ \ then\ Wgpu.next\ =\ Wgpu.curr/2;$$
$$If\ \ |Vgpu.curr-Vgpu.prev|\le Vcpu.prev*\alpha\ ,\ \ then\ Wgpu.next\ =\ Wgpu.curr.$$

PDTS: 首先将一个大任务均分成n个小任务，也即将整个迭代空间均分成n个块，每个处理器前两次计算采用基本块大小，之后每次计算的块大小根据实测性能动态调节。具体来说，每个处理器一旦执行完当前迭代块，根据上一次和当前执行速度来调节下一次计算的工作量，并立即执行下一个迭代块，直到整个迭代空间执行完毕。

报 告 内 容

一、研究背景

二、异构协同并行编程总体框架

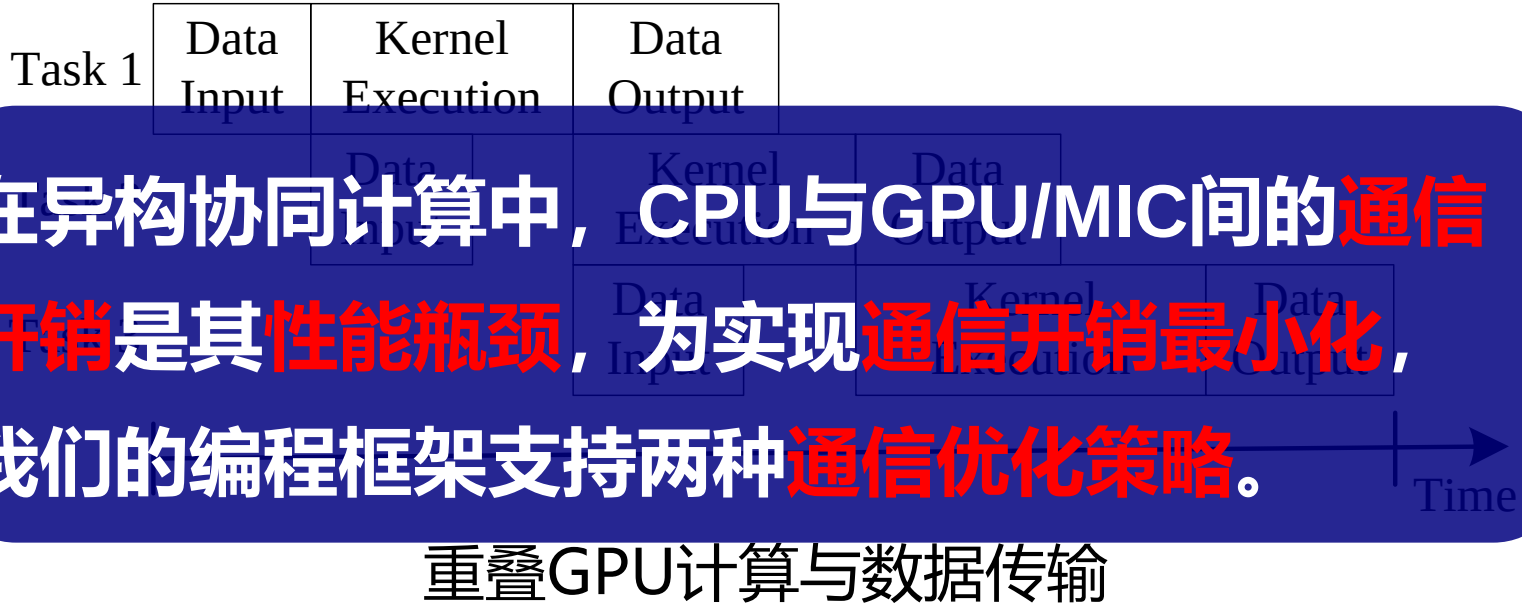
三、异构协同计算中任务调度

四、异构协同计算中通信优化

五、原型实现及应用案例

4. 异构协同计算中通信优化

通信优化策略 1: Pipeline-based Communication Optimization



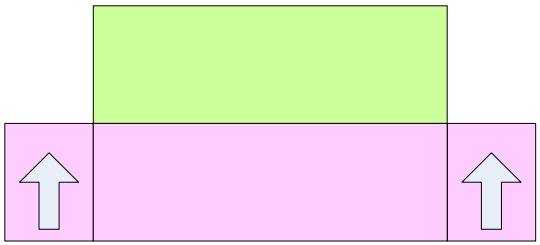
在异构协同计算中，CPU与GPU/MIC间的**通信开销**是其**性能瓶颈**，为实现**通信开销最小化**，我们的编程框架支持两种**通信优化策略**。

重叠GPU计算与数据传输

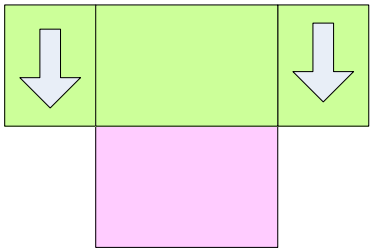
基于流水线的通信优化: 采用CUDA的stream技术和异步传输技术，重叠GPU计算与数据传输，隐藏CPU-GPU间的数据传输时间。

4. 异构协同计算中通信优化

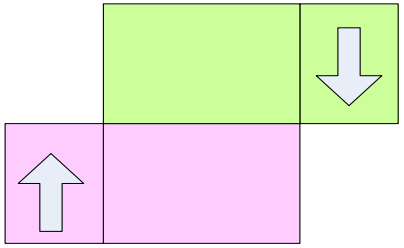
通信优化策略 2: Incremental Data Transfer



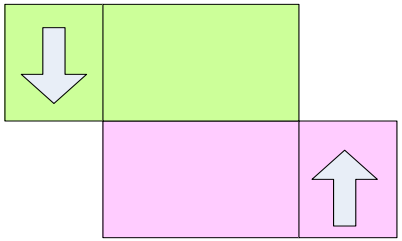
Case 1



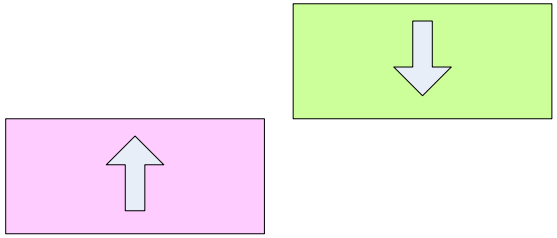
Case 2



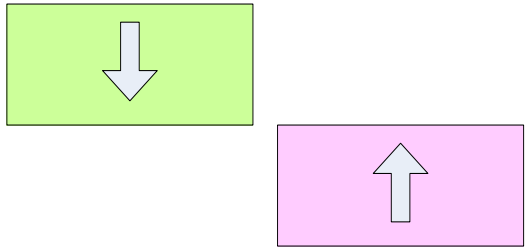
Case 3



Case 4



Case 5



Case 6

增量数据传输: 如果一个计算内核需要循环执行多次，每次都需要在CPU端和GPU/MIC端之间来回传输数据，那么有可能存在重复的不必要的数据传输。我们的增量数据传输策略能够避免重复的不必要的数据传输， 从而减少通信开销。如Case 1所示，左右两端箭头所在部分表示需要将数据从CPU端上传到GPU/MIC端，其余部分表示不需要传输。

报 告 内 容

一、研究背景

二、异构协同并行编程总体框架

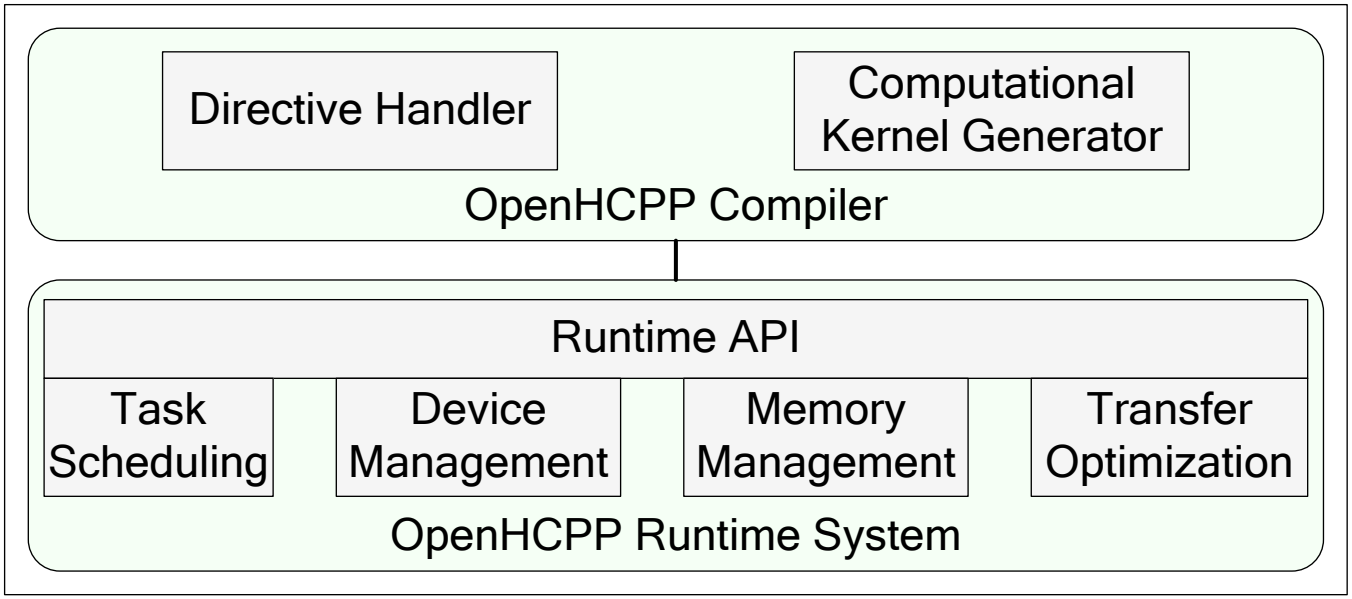
三、异构协同计算中任务调度

四、异构协同计算中通信优化

五、原型实现及应用案例

5. 原型实现及应用案例

OpenHCPP: 一个适应于异构系统的基于指导语句的协同并行编程框架



OpenHCPP 原型系统

源到源的编译器: 指导语句处理器、计算内核生成器

运行时系统: 任务调度、设备管理、存储管理、传输优化

5. 原型实现及应用案例

Table I. Benchmarks used in our experiments

Benchmark	Description	Input Problem Size
Matrix Multiplication (MM)	Compute the product of two matrices	S: 2048×2048 M: 4096×4096 L: 8192×8192 (matrix size)
Jacobi Method (Jacobi)	An algorithm for determining the solutions of a diagonally dominant system of linear equations	S: $10k \times 10k$ M: $20k \times 20k$ L: $20k \times 30k$ (matrix size)
LU Decomposition (LUD)	Factor a matrix as the product of a lower triangular matrix and an upper triangular matrix	S: 6400×6400 M: 9600×9600 L: 12800×12800 (matrix size)
K-means (K-means)	K-means clustering on uniformly distributed data	S: 10×10^6 M: 25×10^6 L: 50×10^6 (random points)
N-body Simulation (N-body)	Approximate the evolution of a system of bodies that continuously interact with each other	S: $256k$ M: $512k$ L: $1024k$ (bodies)
Black-Scholes Option Pricing (BLKS)	Compute option price using Black-Scholes partial differential equation	S: 10×10^6 M: 50×10^6 L: 100×10^6 (options)

测试平台:

1) Two Intel Xeon E5-2640v2 CPU (16 cores at 2.0GHz)

2) A NVIDIA Tesla K20m GPU (2496 CUDA cores at 706MHz)

3) 64GB host memory、5GB device memory

5. 原型实现及应用案例

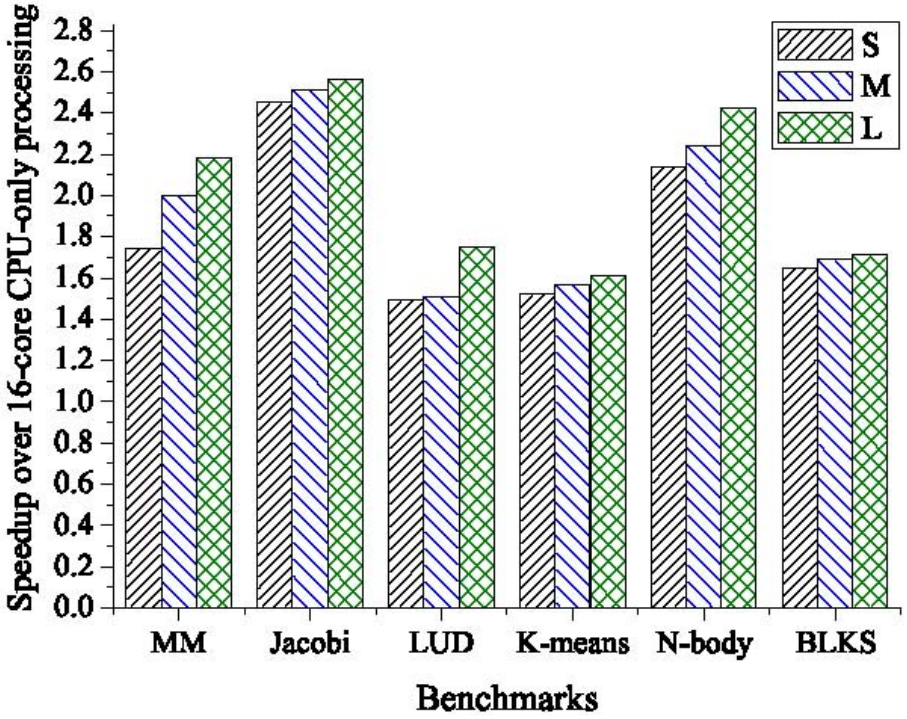


Fig. 1. The speedup of the CPU-GPU co-processing over the CPU-only processing for different problem sizes

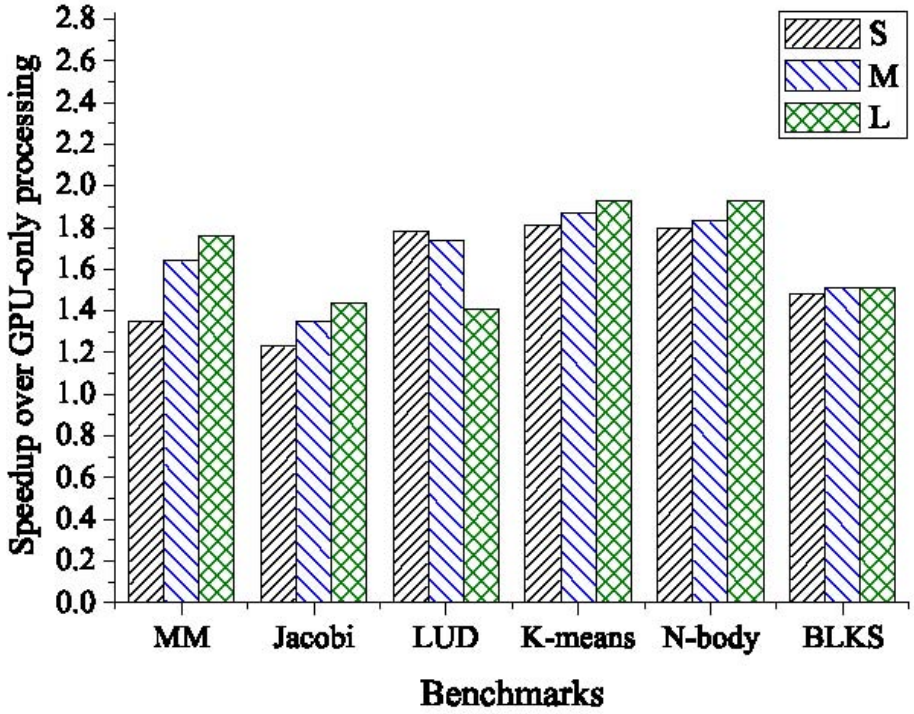


Fig. 2. The speedup of the CPU-GPU co-processing over the GPU-only processing for different problem sizes

图1和图2说明基于我们的编程框架所实现的6个应用案例，使用CPU-GPU协同计算与使用纯CPU或纯GPU计算相比取得了显著的性能提升。


谢谢！