BSPADMM及其在Intel GPU上的移植与优化

2023.10

演讲人: 韩子栋 中国科学院计算机网络信息中心

intel®

1. 问题背景:大规模场景投资组合优化

ADMM是一个将对偶上升法的可分解性和乘子法的上界收敛属性融合在一起的算法

In the classical ADMM form [5], problem (2.2) can be written as

$$\begin{aligned} \min \quad & f(\alpha)+g(z) \\ s.t. \quad & \alpha - z = 0, \end{aligned} \tag{3.1}$$

where $f(\alpha) = \|y - X\alpha\|_2^2$ and $g(z) = \lambda\|z\|_1$. The main steps of ADMM algorithm becomes

$$\alpha^{k+1} = \left(X^T X + \rho I\right)^{-1}\left(X^T y + \rho\left(z^k - u^k\right)\right)$$

$$z^{k+1} = \arg\min_z \lambda\|z\|_1 + \rho\|z - (\alpha^{k+1} + u^k)\|_2^2$$

$$u^{k+1} = u^k + \alpha^{k+1} - z^{k+1}$$

简单来说就是通过求解对偶问题来求解多变量优化问题的快速收敛算法。

加噪图像

去噪图像

ADMM算法在图像处理，语音降噪等领域应用广泛

在其他领域如价格预测也被广泛使用

2. 算法优化

目标函数：

$$\hat{\boldsymbol{\alpha}} = L(\boldsymbol{\alpha}, \lambda) = \arg\min_{\boldsymbol{\alpha}} \| \boldsymbol{y} - X\boldsymbol{\alpha} \|_2^2 + \lambda \| \boldsymbol{\alpha} \|_p^p \qquad \text{s.t} \quad \boldsymbol{z} = \boldsymbol{y} - X\boldsymbol{\alpha}$$

目标函数变换：

$$\min_{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N} \sum_{i=1}^{N} f_i(\mathbf{x}_i) \quad \text{s.t.} \sum_{i=1}^{N} A_i\mathbf{x}_i = c,$$

ADMM算法
迭代步骤

$$\begin{aligned} x^{k+1} &:= (A^TA + \rho I)^{-1}\left(A^Tb + \rho(z^k - u^k)\right) \\ z^{k+1} &:= S_{\lambda/\rho}\left(x^{k+1} + u^k\right) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

N block ADMM(2020)：让算法更好并行

$$\begin{aligned} \arg\min_{\alpha} \quad & \sum_{i=1}^{N} \|\alpha_i\|_1 \\ \text{s.t.} \quad & \sum_{i=1}^{N} X_i\alpha_i = y \end{aligned} \qquad (3.2)$$

改进一、目标函数变量分离

$$\begin{aligned} \alpha^{k+1} &\leftarrow \arg\min_{\alpha_i} \left\{\|\alpha_i\|_1 + \frac{\rho}{2}\|X_i\alpha_i + c_i\|_2^2\right\}, \\ c_i^k &= \sum_{j\neq i} X_j\alpha_j^k - y - \frac{u^k}{\rho}, \end{aligned} \qquad (3.3)$$

改进二、改进迭代步骤

2. BSPADMM算法

BSPADMM变换目标函数为求解下列子问题

近端项

$$\arg\min_{\alpha_i}\{\|\alpha_i\|_1+\frac{\rho}{2}\|X_i\alpha_i+c_i\|_2^2+\boxed{\frac{1}{2}\left\|\alpha_i-\alpha_i^k\right\|_{P_i}^2}\},$$

迭代步骤也变为

$$\begin{aligned}\alpha_i^{k+1}&=\arg\min_{\alpha_i}\|\alpha_i\|_1+\frac{\rho}{2}\|X_i\alpha_i+\sum_{j\neq i}X_j\alpha_j^k-c-\frac{\lambda^k}{\rho}\|^2\\&+\frac{1}{2}\|\alpha_i-\alpha_i^k\|_{P_i}^2\\&=\arg\min_{\alpha_i}\|\alpha_i\|_1+\left\langle\rho A_i^{\top}\left(A\alpha^k-c-\frac{\lambda^k}{\rho}\right),\alpha_i\right\rangle\\&+\frac{1}{2}\left\|\alpha_i-\alpha_i^k\right\|_2^2\end{aligned}\tag{3.4}$$

1.这种变换非常有利于并行
2.子问题的最小化可以很容易地通过收缩操作来计算，而非矩阵逆计算。
3.每个计算节点都可以在本地计算步长参数p

$$Shrink\_l1(x,r)=\mathrm{sgn}(x)\max\{abs(x)-r,0\},$$

收缩操作

2. BSPADMM算法

BSPADMM的迭代步骤

$$\begin{aligned}\alpha_i^{k+1} &= \arg\min_{\alpha_i}\|\alpha_i\|_1 + \frac{\rho}{2}\|X_i\alpha_i + \sum_{j\neq i} X_j\alpha_j^k - c - \frac{\lambda^k}{\rho}\|^2 \\ &+ \frac{1}{2}\|\alpha_i - \alpha_i^k\|_{P_i}^2 \\ &= \arg\min_{\alpha_i}\|\alpha_i\|_1 + \left\langle \underbrace{\rho}_{\text{步长}} A_i^T\left(A\alpha^k - c - \frac{\lambda^k}{\rho}\right), \alpha_i\right\rangle \\ &+ \frac{1}{2}\left\|\alpha_i - \alpha_i^k\right\|_2^2 \qquad (3.4)\end{aligned}$$

加快算法收敛速度，采用动态的步长

$$G = \|\mathbf{r} - \mathbf{r}_t\|_2^2 + \|Ax - Axt\|_2^2 + \|\mathbf{r}_t\|_2^2$$

$$\rho = 2\mathbf{r}_t^T(\mathbf{r}_t - \mathbf{r})/G + 1;$$

在此基础上，令每一个处理器有自己的步长

$$G_i = \|\mathbf{r} - \mathbf{r}_t\|_2^2 + \|Ax_i - Axt_i\|_2^2 + \|\mathbf{r}_t\|_2^2$$

$$\rho_i = 2\mathbf{r}_t^T(\mathbf{r}_t - \mathbf{r})/G_i + 1;$$

Algorithm 2 BSPADMM

input: dataset X, y, ε
output: the sparse representation α
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &i);
$X_i \leftarrow readCSC(\text{file}, i)$
$(m_i, n_i) \leftarrow size(X_i)$
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, n_l$ **do**
    $c \leftarrow u^k/\rho_i + r + Ax_i(:, j)$
    $x_j \leftarrow 1/(\rho_i c^T A_i(:, j))$
    $\hat{\alpha}_i^k(j) \leftarrow Shrink\_l1(x_j, 1/\rho_i)$
    $Axt_i(:, j) \leftarrow \hat{\alpha}_j X_i(:, j)$
    $Ax_i(:, j) \leftarrow \alpha_j X_i(:, j)$
  end for
  $Axrt_i \leftarrow rowsum(map(Axt_i))$
  $Axr_i \leftarrow rowsum(map(Ax_i))$
  $r_t \leftarrow y - sum(Axrt_i)$// sum by MPI_Allreduce
  $r \leftarrow y - sum(Axr_i)$
  if $\|r\|_2 < \varepsilon$ **then**
    break
  end if
  $\alpha_i^{k+1} \leftarrow (1 - \rho_i)\alpha_i^k + \rho_i\hat{\alpha}_i^k$
  $u^{k+1} \leftarrow u^k + r_t$
  $\rho_i \leftarrow update(Ax_i, Axt_i, r, r_t)$
end for

BSPADMM的完整算法流程

发表论文:BSPADMM: Block Splitting Proximal ADMM for Sparse Representation with Strong Scalability

计算流程

数据预处理
SliceData
CNICSparseMatrix
readSparseData
sortCoo2CSR

内循环
conjugate_gradient
shrink

并行计算行和
mkl_sparse_d_create_csr
mkl_sparse_d_mv
mkl_sparse_d_update_values

残差计算与更新

Algorithm 2 BSPADMM

input: dataset X, y, ε
output: the sparse representation α
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &i);
$X_i \leftarrow readCSC(\text{file}, i)$
$(m_i, n_i) \leftarrow size(X_i)$
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, n_l$ **do**
    $c \leftarrow u^k/\rho_i + r + Ax_i(:, j)$
    $x_j \leftarrow 1/(\rho_i c^T A_i(:, j))$
    $\hat{\alpha}_i^k(j) \leftarrow Shrink\_l1(x_j, 1/\rho_i)$
    $Axt_i(:, j) \leftarrow \hat{\alpha}_j X_i(:, j)$
    $Ax_i(:, j) \leftarrow \alpha_j X_i(:, j)$
  end for
  $Axrt_i \leftarrow rowsum(map(Axt_i))$
  $Axr_i \leftarrow rowsum(map(Ax_i))$
  $r_t \leftarrow y - sum(Axrt_i)$// sum by MPI_Allreduce
  $r \leftarrow y - sum(Axr_i)$
  if $\|r\|_2 < \varepsilon$ **then**
    break
  end if
  $\alpha_i^{k+1} \leftarrow (1 - \rho_i)\alpha_i^k + \rho_i \hat{\alpha}_i^k$
  $u^{k+1} \leftarrow u^k + r_t$
  $\rho_i \leftarrow update(Ax_i, Axt_i, r, r_t)$
end for

计算问题

稀疏矩阵格式转换
矩阵预处理

稀疏矩阵乘
矩阵向量乘

非矩阵独立向量
非矩阵依赖向量

数据拷贝
数据同步

Time	Price	Volume	Total Amount
2023/1/11 9:30:00	35.78	508	8026400
2023/1/11 9:30:01	35.76	745	11771000
2023/1/11 9:30:02	35.80	782	12355600
2023/1/11 9:30:03	35.82	275	4345000
2023/1/11 9:30:04	35.80	432	6825600
2023/1/11 9:30:05	35.80	546	8626800
2023/1/11 9:30:06	35.81	545	8611000
2023/1/11 9:30:07	35.78	487	7694600
2023/1/11 9:30:08	35.79	605	9559000
2023/1/11 9:30:09	35.77	556	8784800
2023/1/11 9:30:10	35.82	375	5925000
2023/1/11 9:30:11	35.79	728	11502400
2023/1/11 9:30:12	35.80	400	6320000
2023/1/11 9:30:13	35.80	200	3160000
2023/1/11 9:30:14	35.84	1100	17380000
2023/1/11 9:30:15	35.88	937	14804600

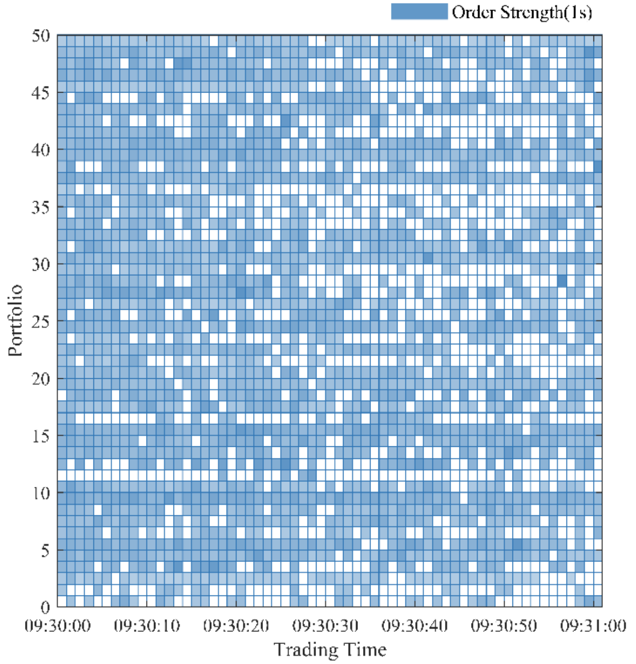
数据来源于真实交易数据

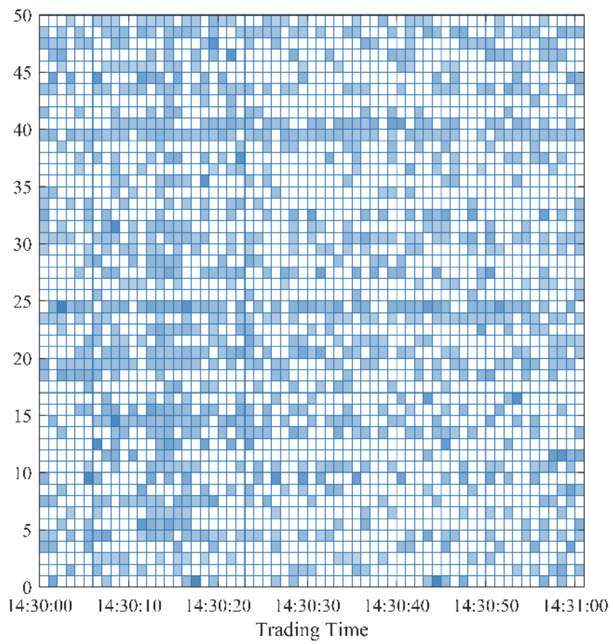
Dataset	Sparsity	Feature dimension	Number of signals
I		8 ~ 128	$2^{11} \sim 2^{15}$
II	0.05	8 ~ 64	$2^{16} \sim 2^{18}$
III		8 ~ 16	2^{24}

变量可分离的目标函数

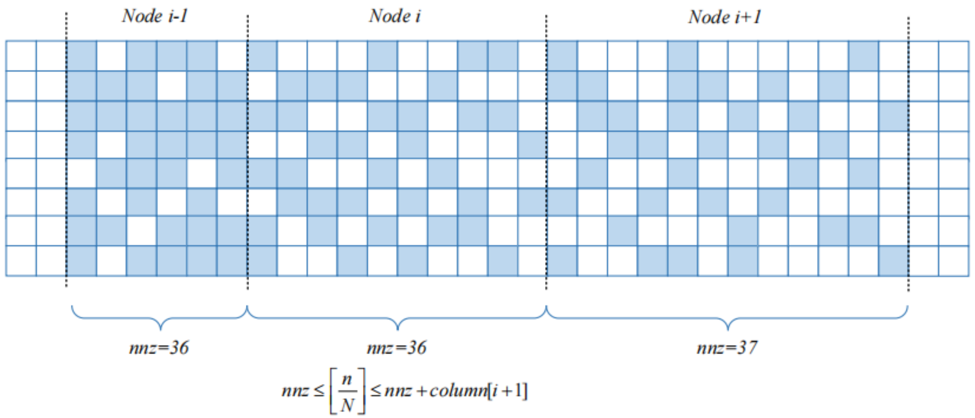
$$\min_{\tilde{\boldsymbol{x}}\in \boldsymbol{R}^{m+n},\, \boldsymbol{z}\in \boldsymbol{R}^{m+n}} \tilde{\boldsymbol{x}}'\tilde{\Sigma}\tilde{\boldsymbol{x}} - \tilde{E}(R)'\tilde{\boldsymbol{x}} + \sum_{i=1} f_i(z_i) + \mathrm{I}_{C_1}(\tilde{\boldsymbol{x}}) + \mathrm{I}_{C_2}(\boldsymbol{z})$$

$$\text{s.t.}\quad \tilde{x}_i - z_i = 0,\ i = 1,2,\ldots,n+m$$





不同时间段数据密度不同



数据量比较大，需要切分多节点完成

通过较为合理的切分来保证负载均衡

2. BSPADMM算法

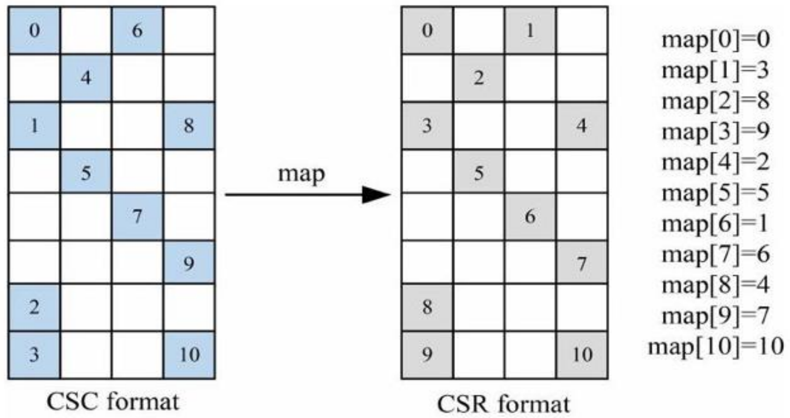
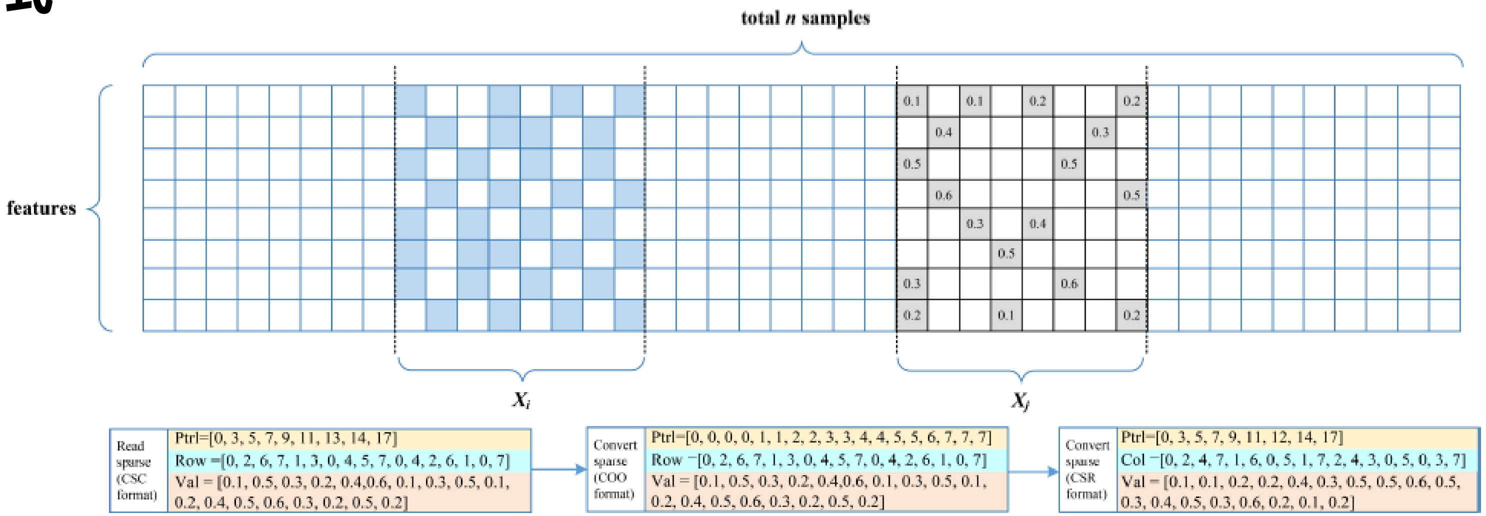


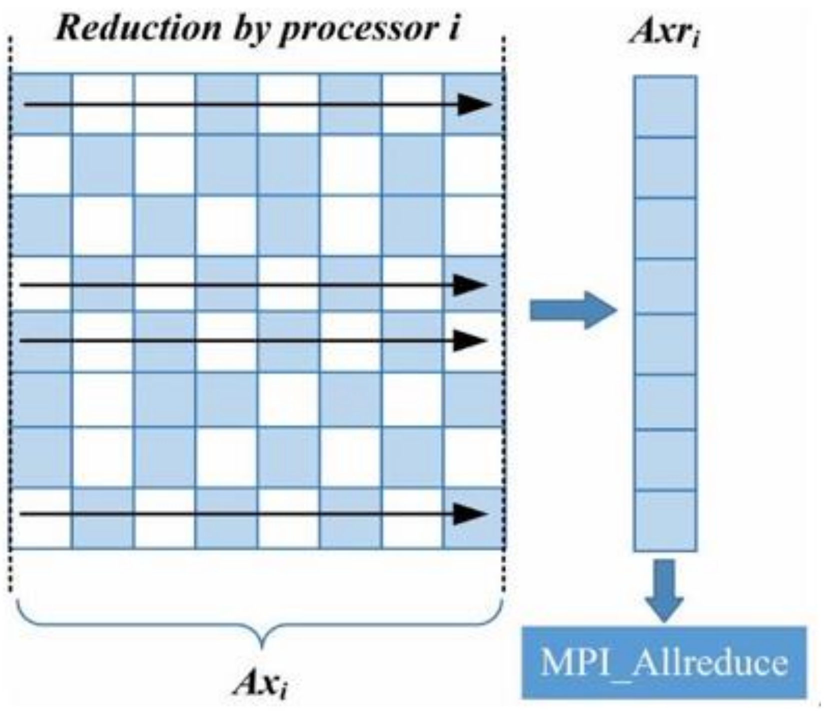
Table 1: Mapping the CSC matrix to the CSR matrix in parallel.

```
void map(int nnz, int* map,double* Ax_mapped,
    double* Ax){
    // map the value of the CSC  to CSR matrix
    #pragma omp parallel for
    for (int i = 0; i < nnz; ++i){
     int j = map[i];
     Ax_mapped[j] = Ax[i];
    }
}
```

在迭代过程中，如果将稀疏矩阵存储为压缩稀疏行（CSR）格式，则计算列向量乘积是非常耗时的。BSPADMM按压缩稀疏列(CSC)格式计算。通过一个整数指针将该值CSC矩阵映射到CSR格式



2 BSPADMM算法




```
sparse_matrix_t csrA;
matrix_descr descrA;
descrA.type = SPARSE_MATRIX_TYPE_GENERAL;
mkl_sparse_d_create_csr(&csrA,
    SPARSE_INDEX_BASE_ZERO, m, n,&csrptrl[0], &
    csrptrl[1], &csrcol[0], Ax_mapped);
mkl_sparse_optimize(csrA);
for (int iter = 0; iter < maxit; ++iter){
  //  iteration
  ...
  //  map the CSC matrix to the CSR matrix
  ...
  //  local matrix reduction
          mkl_sparse_d_update_values(csrA,0,
              NULL,NULL,Ax_mapped);
          mkl_sparse_d_mv(
              SPARSE_OPERATION_NON_TRANSPOSE,
              1.0, csrA, descrA, onesvector,
              0.0, Axr);

  //  global matrix reduction for computing the
      residual.
  MPI_Allreduce(Axr,Axrsum,n,MPI_DOUBLE,MPI_SUM,
      MPI_COMM_WORLD);
  ...
}
```

每个处理器将矩阵的行和计算成一个向量。函数MPI_ALLreduce将所有的向量加在一起。

2. BSPADMM算法

优化方法

算法优化

- 添加近端项
- 自适应步长

→ 加快收敛 提升可扩展性

并行优化

- 并行I/O
- 并行稀疏SPMV
- 矩阵格式映射
- 数据负载均衡

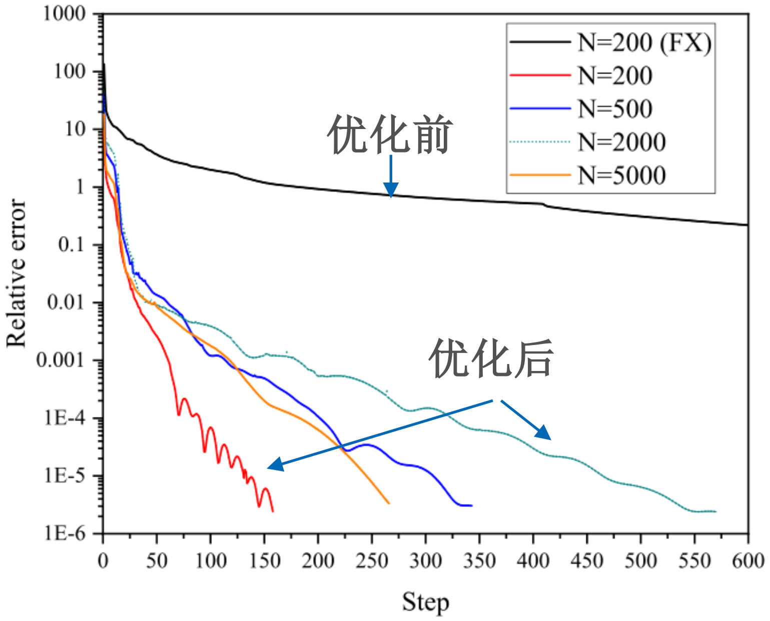
→ 提升效率

适用于所有硬件的逻辑层面优化

3. 面向INTEL GPU的移植与优化

首先在CPU与Nvdia GPU上进行实现 使用了 oneAPI MKL

证明算法的正确性与具有加速效果



与原算法收敛性对比

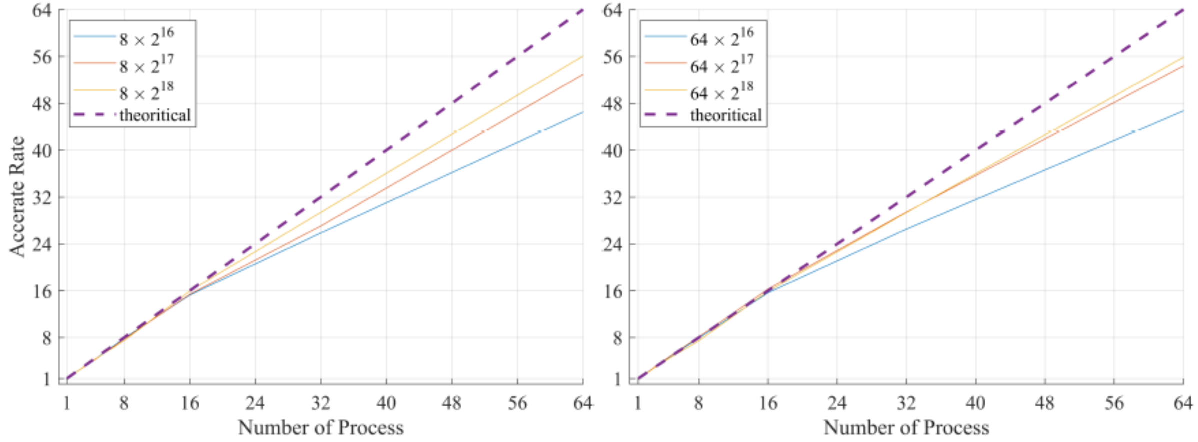
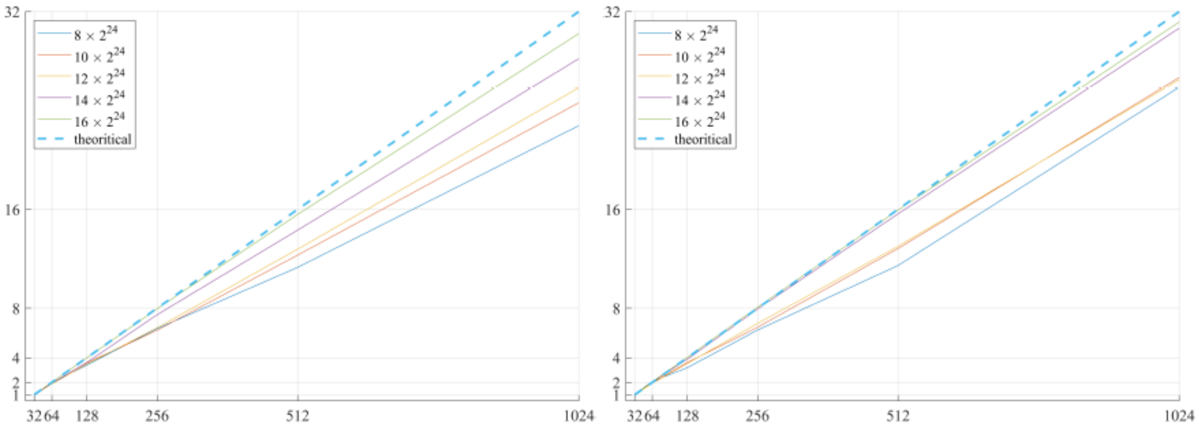


Figure 5. Run tests for the dataset II. The number of samples n of X ranges from 2^{16} to 2^{18}.



千核加速比与运行效率

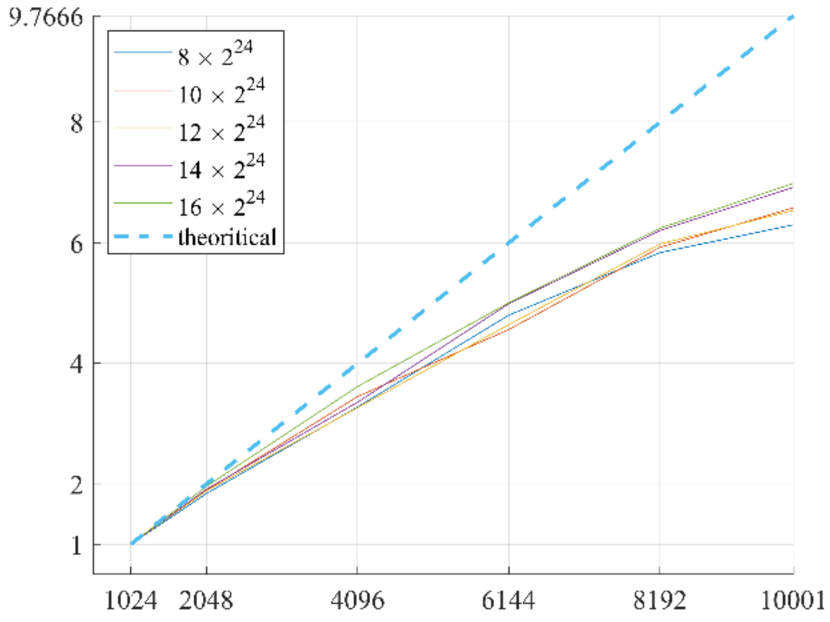
3. 面向INTEL GPU的移植与优化

使用 Intel oneAPI Math Kernel Library

```
#ifdef USE_NEW_MKL
            mkl_sparse_d_update_values(csrA,0,NULL,NULL,sp_ax_data.Ax_mapped);

            mkl_sparse_d_mv(SPARSE_OPERATION_NON_TRANSPOSE, 1.0, csrA, descrA,
                            sp_ax_data.onesvector , 0.0, sp_ax_data.Axreduction);

            mkl_sparse_d_update_values(csrB,0,NULL,NULL,sp_ax_data.Axt_mapped);

            mkl_sparse_d_mv(SPARSE_OPERATION_NON_TRANSPOSE, 1.0, csrB, descrA,
                            sp_ax_data.onesvector , 0.0, sp_ax_data.Axtreduction);

#else

        mkl_dcsrmv(Nchar,&local_rows,&local_cols,
                   &one,matdescra,sp_ax_data.Ax_mapped,csrcol.data(),
                   &csrptrl[0],&csrptrl[1],sp_ax_data.onesvector,&zero,sp_ax_data.Axreduction);

        mkl_dcsrmv(Nchar,&local_rows,&local_cols,
                   &one,matdescra,sp_ax_data.Axt_mapped,csrcol.data(),
                   &csrptrl[0],&csrptrl[1],sp_ax_data.onesvector,&zero,sp_ax_data.Axtreduction);

#endif
```

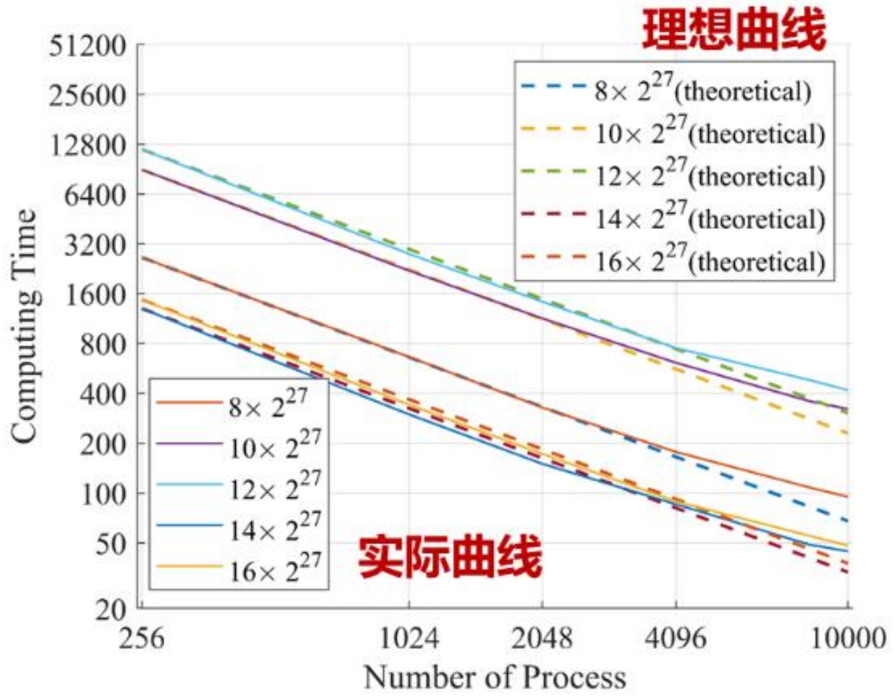
3. 面向INTEL GPU的移植与优化

在万核Intel CPU集群的并行效率



万核加速比和并行效率

	128	1024	2048	4096	8192	10000
8 * 2^{27}	100.00%	92.08%	81.93%	80.01%	72.96%	64.49%
10 * 2^{27}	100.00%	94.80%	86.25%	76.02%	74.08%	67.38%
12 * 2^{27}	100.00%	94.34%	81.65%	77.39%	74.75%	66.94%
14 * 2^{27}	**100.00%**	**95.59%**	**83.92%**	**83.17%**	**77.55%**	**70.86%**
16 * 2^{27}	**100.00%**	**98.03%**	**90.41%**	**83.42%**	**78.03%**	**71.53%**

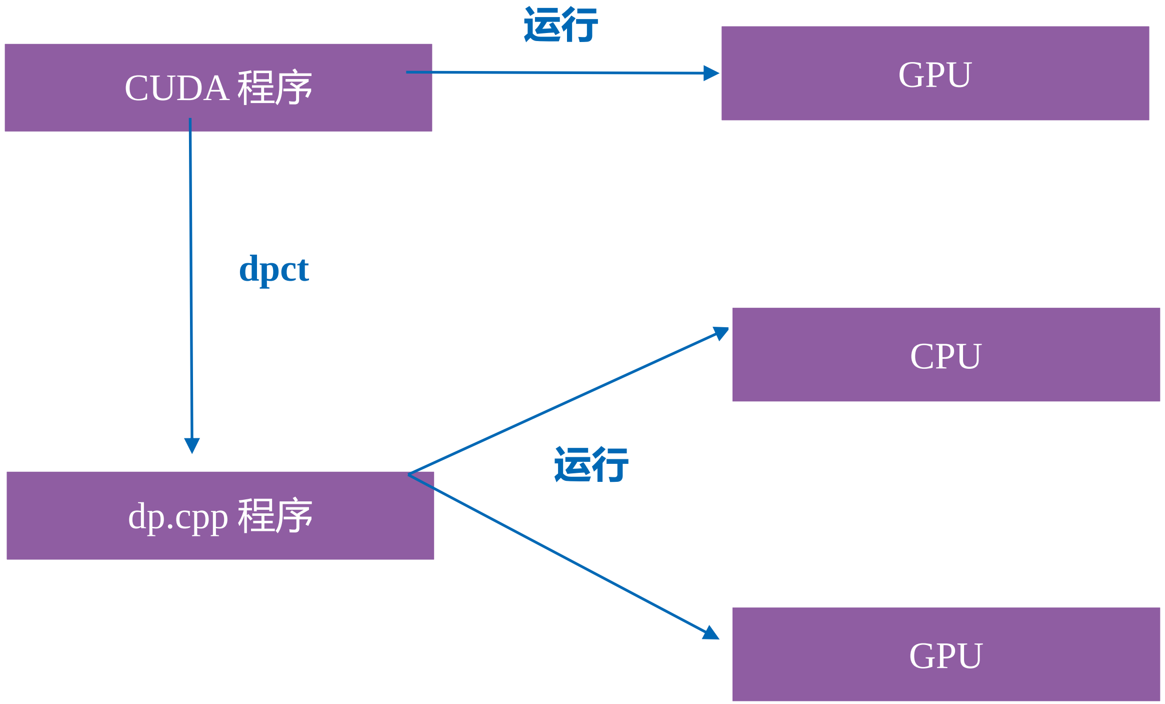


Cores	ARADMM		N-ADMM		BSPADMM	
	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
32	8.2×10^4	1.23×10^5	1.62×10^5	1.81×10^5	1.46×10^5	1.11×10^5
64		9.81×10^4		1.23×10^5	1.45×10^5	5601.84
128		8.21×10^4		8.31×10^4	1.41×10^5	2836.22
256		7.05×10^4		5.92×10^4	1.41×10^5	1410.73
512		6.32×10^4		3.94×10^4	1.40×10^5	715.42
1024		5.24×10^4		3.03×10^4	1.40×10^5	366.18

与已有方法比较

3. 面向INTEL GPU的移植与优化

从CUDA向DPC++的迁移



移植示例：

c2s --in-root=. src/<code>.cu --cuda-include-path=<path>/include

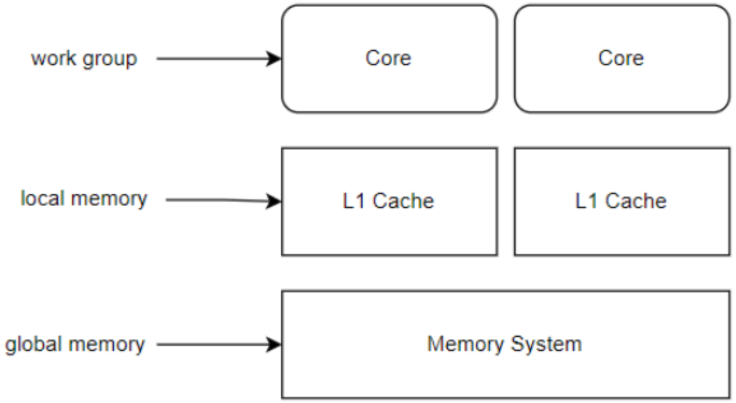
编程模型

CUDA	thread	wrap	block	grid
DPC++(oneAPI)	work item	sub group	work group	nd range
OpenCL	work item	sub group	work group	nd range

数据管理模型

CUDA	shared	Unified Memory	_syncthreads
DPC++(oneAPI)	local	Unified Shared Memory	barrier
OpenCL	local	Unified Shared Memory	barrier

oneAPI在CPU上的映射



3. 面向INTEL GPU的移植与优化

CUDA 程序

```
__global__ void updateXnew(double* xOld, double*dOld,double*xNew, double alpha, int nn){
    // xNew = xOld  + alpha.*dOld;
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    if (x>=nn)
        return;
    xNew[x] = xOld[x] + alpha * dOld[x];
    xOld[x] = xNew[x];


}

__global__ void update_rew(double* rnew,double *b, int nn){
    // xNew = xOld  + alpha.*dOld;
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    if (x>=nn)
        return;
    rnew[x] = b[x] - rnew[x];


}
```

c2s工具转换的DPC++程序

```
void updateXnew(double* xOld, double*dOld,double*xNew, double alpha, int nn,
                sycl::nd_item<3> item_ct1){
    // xNew = xOld  + alpha.*dOld;
    int x = item_ct1.get_group(2) * item_ct1.get_local_range().get(2) +
            item_ct1.get_local_id(2);
    if (x>=nn)
        return;
    xNew[x] = xOld[x] + alpha * dOld[x];
    xOld[x] = xNew[x];


}

void updateXnew_(double* xOld, double*dOld,double*xNew, double *alpha, int nn,
                 sycl::nd_item<3> item_ct1){
    // xNew = xOld  + alpha.*dOld;
    int x = item_ct1.get_group(2) * item_ct1.get_local_range().get(2) +
            item_ct1.get_local_id(2);
    if (x>=nn)
        return;
    xNew[x] = xOld[x] + (*alpha) * dOld[x];
    xOld[x] = xNew[x];


}
```

oneAPI工具易用、界面友好、性能保障；OpenCL风格 易于阅读，便于维护，减轻开发者工作。DPCPP语言会传入迭代器item来帮助对每个线程的操控

直接利用移植工具
除了一些部分没有放在GPU，几乎是可以直接运行在INTEL GPU上

```
double  res_temp_ptr_ct7 = &bNorm;
if (sycl::get_pointer_type(&bNorm, handle->get_context()) !=
        sycl::usm::alloc::device &&
    sycl::get_pointer_type(&bNorm, handle->get_context()) !=
        sycl::usm::alloc::shared) {
    res_temp_ptr_ct7 =
        sycl::malloc_shared<double>(1, dpct::get_default_queue());
}
oneapi::mkl::blas::nrm2(*handle, n, b, 1, res_temp_ptr_ct7);
if (sycl::get_pointer_type(&bNorm, handle->get_context()) !=
        sycl::usm::alloc::device &&
    sycl::get_pointer_type(&bNorm, handle->get_context()) !=
        sycl::usm::alloc::shared) {
    handle->wait();
    bNorm = *res_temp_ptr_ct7;
    sycl::free(res_temp_ptr_ct7, dpct::get_default_queue());
} // bNorm;
```

使用oneapi自带的工具对CUDA程序进行转换后

在调用核函数及自带的库函数时，会自动添加对变量类型检测的if语句

根据结果可能会创建一个share变量来保证程序的正确运行

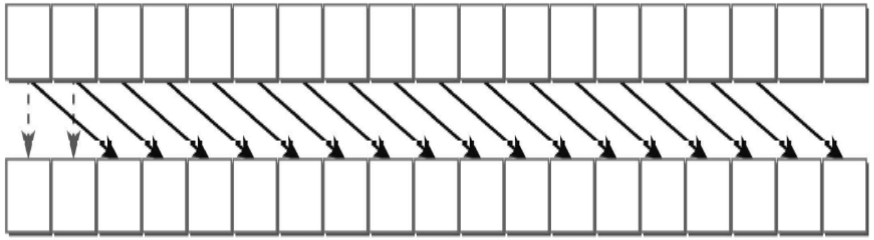
为了程序运行的速度，我们在声明变量类型后将这些语句去掉了

但这个设计本身保证了移植的成功率

3. 面向INTEL GPU的移植与优化

DPC++对 __shfl_down_的处理

```
inline double __shfl_down_(double var, unsigned int srcLane,
                           sycl::nd_item<3> item_ct1, int width=WARP_) {
  sycl::int2 a = *reinterpret_cast<sycl::int2 *>(&var);
  a.x() = dpct::shift_sub_group_left(item_ct1.get_sub_group(), a.x(), srcLane,
                                     width);
  a.y() = dpct::shift_sub_group_left(item_ct1.get_sub_group(), a.y(), srcLane,
                                     width);
  return *reinterpret_cast<double*>(&a);
}
```



_shfl_up (val,2): 将值转移到右边两个通道中

DPC++自带类 模板 STL库 不需要使用第三方库

```
this->csr_row_ptrl = dpct::device_vector<int>(coo_matrix->n_rows + 1);

auto thrust_raw_pointer_cast_A_cudata_ct0 =
    dpct::get_raw_pointer(A->cudata.data());
auto thrust_raw_pointer_cast_A_csr_row_ptrl_data_ct1 =
    dpct::get_raw_pointer(A->csr_row_ptrl.data());
auto thrust_raw_pointer_cast_A_cucol_data_ct2 =
    dpct::get_raw_pointer(A->cucol.data());
auto A_n_rows_ct5 = A->n_rows;
```

DPC++不仅对CUDA一些接口进行实现，同时也更贴合熟悉C++的编程者

3. 面向INTEL GPU的移植与优化

DPC++使用提交形式执行核函数与数据拷贝
隐式的在主机和设备之间异步传输数据与执行

```
q_ct1.parallel_for(sycl::nd_range<3>(block * sycl::range<3>(1, 1, 128),
                                     sycl::range<3>(1, 1, 128)),
                   [=](sycl::nd_item<3> item_ct1) {
                     update_rew(rOld, b, n, item_ct1);
                   });

// dOld  ;
q_ct1.memcpy(dOld, rOld, size_x).wait();
```

DPC++使用UMS进行内存管理
允许内存在主机和设备之间共享，以减少数据传输开销
数据的迁移和分配是隐式完成的

```
void CSRMatrix::print_matrix(){

//    for (int i = 0 ; i < this->n_element; ++i){
//        printf("%.4f\n",this->cucol[i]);
//    }
    show_res_T<double>((double *)dpct::get_raw_pointer(this->cudata.data()),
                       this->n_element);
}
```

3. 面向INTEL GPU的移植与优化

DPC++和整个oneAPI的生态是兼容的

直接编程	Intel® oneAPI DPC++/C++ Compiler (LLVM)	Y
	Intel® C++ Compiler Classic	Y
	Intel® Fortran Compiler (LLVM)	
	Intel® Fortran Compiler Classic	
	Intel® FPGA Add-on for oneAPI Base Toolkit	
分析工具	Intel® VTune™ Profiler	Y
	Intel® Advisor	
	Intel® Inspector	
	Intel® Trace Analyzer & Collector	
	Intel® Cluster Checker	
	Intel® Distribution for GDB	Y
基于 API 的编程	Intel® MPI Library	Y
	Intel® oneAPI DPC++ Library	Y
	Intel® oneAPI Math Kernel Library	
	Intel® oneAPI Data Analytics Library	
	Intel® oneAPI Threading Building Blocks	
	Intel® oneAPI Video Processing Library	
	Intel® oneAPI Collective Communications Library	
	Intel® oneAPI Deep Neural Network Library	
	Intel® Integrated Performance Primitives	Y

过程中也使用了很多oneAPI的工具

```
u02@pc:~/multistageALM/sycl$ gdb-oneapi ./ADMM_Solver_sycl
GNU gdb (Intel(R) Distribution for GDB* 2022.2) 11.2
Copyright (C) 2022 Free Software Foundation, Inc.; (C) 2022 Intel Corp.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.

For information about how to find Technical Support, Product Updates,
User Forums, FAQs, tips and tricks, and other support information, please visit:
<http://www.intel.com/software/products/support/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ADMM_Solver_sycl...
(gdb) n
The program is not being run.
(gdb)
```

```
// optimize : combine the mpi reduce
MPI_Allreduce(sp_ax_data.Axtreduction,dualdata.global_res_t,
2 * n_global + 1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
cblas_dcopy(n_global,globalb.val.data(),1,dualdata.res,1);
cblas_dcopy(n_global,globalb.val.data(),1,dualdata.res_t,1);
cblas_daxpy(n_global,-1.0,dualdata.global_res,1,dualdata.res,1);
cblas_daxpy(n_global,-1.0,dualdata.global_res_t,1,dualdata.res_t,1);
```

3. 面向INTEL GPU的移植与优化 使用 Intel VTune Profiler

热点分析后我们发现只有一些库函数明显慢于cublas

```
oneapi::mkl::blas::nrm2(*handle, n, b, 1, res_temp_ptr_ct7);
```

手写的库函数

```
q_ct1.submit([&](sycl::handler &cgh) {
    cgh.parallel_for(
        sycl::nd_range<3>(block * sycl::range<3>(1, 1, 512),
                          sycl::range<3>(1, 1, 512)),
        sycl::reduction(res_temp_ptr_ct177, std::plus<>()),
        [=](sycl::nd_item<3> item_ct1, auto& res_temp_ptr_ct177){
            int x = item_ct1.get_group(2) * item_ct1.get_local_range().get(2) +
            item_ct1.get_local_id(2);
            if (x<n){
                res_temp_ptr_ct177 += rNew[x] * rNew[x];
            }
    });
}).wait();
```

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically

Function	Module	CP
Intel::OpenCL::CPUDevice::AffinitizeThreads::ExecuteIteration	libcpu_device_emu.so.2022.13.3.0	
Intel::OpenCL::Utils::AtomicCounter::operator long	libcpu_device_emu.so.2022.13.3.0	
cl::sycl::queue::submit_impl	libsycl.so.5	
Intel::OpenCL::Utils::AtomicCounter::operator long	libcpu_device.so.2022.13.3.0	
Intel::OpenCL::CPUDevice::AffinitizeThreads::ExecuteIteration	libcpu_device.so.2022.13.3.0	
[Others]	N/A*	

*N/A is applied to non-summable metrics.

我们也尝试了用其他的库函数进行替代

3. 面向INTEL GPU的移植与优化

```
q_ct1.submit([&](sycl::handler &cgh) {
    cgh.parallel_for(
        sycl::nd_range<3>(block * sycl::range<3>(1, 1, 512),
                          sycl::range<3>(1, 1, 512)),
        sycl::reduction(res_temp_ptr_ct177, std::plus<>()),
        [=](sycl::nd_item<3> item_ct1, auto& res_temp_ptr_ct177){
            int x = item_ct1.get_group(2) * item_ct1.get_local_range().get(2) +
            item_ct1.get_local_id(2);
            if (x<n){
                res_temp_ptr_ct177 += rNew[x] * rNew[x];
            }
    });
}).wait();
```

在与CUDA程序对比过程中，我们进一步的发现提供的Reduce加速比慢于预期

从官方论坛查询的一些类似问题，我们认为是编译器的问题

向工程师提交以后也得到了验证，这个问题最终在2023.2版本中得到了修复

现在速度不慢于在CUDA版本的运行速度

intel®