rho和H小kernel优化经验

用Fortran写从核函数

用Fortran写从核函数和用c写从核函数最大的不同点是传递数组的方式(下面的代码中matrix数组的传递)，Fortran从核函数使用的cray指针，编译时需要加上-fcray-pointer参数。

```
!主核代码
program main
    type :: param_t
        integer*8 :: matrix_loc
        integer m,n
        real*8 :: k
    end type param_t
    type(param_t) param
    integer,external :: slave_func
    real*8 matrix(100)
    integer :: m,n
    real*8 :: k
    param%matrix_loc=loc(matrix)
    param%m=m
    param%n=n
    param%k=k
    call athread_spawn(slave_func, param)
    call athread_join()
end program main
```

```
!从核代码
subroutine func(param)
    type :: param_t
        integer*8 :: matrix_loc
        integer m,n
        real*8 :: k
    end type param_t
    type(param_t) :: param,local_param
    real*8 gl_matrix(100)
    pointer(matrix_ptr,gl_matrix)
    integer i
    call crts_dma_get(local_param,param,24)
    matrix_ptr=local_param%matrix_loc
    do i=1,100,1
        print *,gl_matrix(i)
    end do
end subroutine func
```

rho的优化

计算rho的流程(c语言描述)

```
for(int i_my_batch=1;i_my_batch<n_my_batches_work;i_my_batch++)
{
    tab_atom_centered_coords();
    prune_density_matrix_sparse_polar_reduce_memory();
    collect_batch_centers_p2();
    for(int i_index=0;i_index<batches_work_size(i_my_batch);i_index++)
    {
        prune_radial_basis_p2();
        tab_local_geometry_p2();
        tab_trigonom_p0();
        tab_wave_ylm_p0();
        evaluate_radial_functions_p0();
        evaluate_waves_p2();
    }
    dgemm();
    matrix_ddot();
}
```

热点分析

	tab_atom	prune_density	i_index_loop	dgemm	matrix_ddot	collect_batch
占比	31%	25%	31%	12%	1%	0.2%
时钟周期数	21亿	17亿	21亿	8亿	0.7亿	0.17亿

tab_atom_centered_coords优化

```
for(int i=0;i<n_centers_integrals;i++)
{
    dir_tab[0,i]=coord_current[0]-coords_center[0,centers_basis_integrals[i]];
    dir_tab[1,i]=coord_current[1]-coords_center[1,centers_basis_integrals[i]];
    dir_tab[2,i]=coord_current[2]-coords_center[2,centers_basis_integrals[i]];
}
```

1. 打表减少离散访存。tab_atom_centered_coords这个函数需要运行n_my_batches_work次，也就是最外层循环的次数，每一次运行对coords_center[1,centers_basis_integrals[i]]的访问都是一样的，所以可以打个表，这样可以消除对centers_basis_integrals这个数组的访问，同时把对coords_center的离散访存转换成了连续的访存。
2. 通过openacc上从核。

优化前	优化后	加速比
21亿	4亿	5.25

prune_density_matrix_sparse_polar_reduce_memory优化

1. 这个函数在H中也有一个类似的函数，但是H中这个类似的函数上从核后的加速比是12.5。原因是rho中的这个函数百分之六十以上的时间都花在了离散访存上，尝试用过非阻塞的跨步dma，但是调用相应函数后出了bug，还没有解决。

优化前	优化后	加速比
17亿	9.8亿	1.7

matrix_ddot优化

1. simd。
2. double buffer，一开始使用的是非阻塞的dma，但是最后的结果错了，说明dma还没完成计算就已经做完了，所以只能用阻塞的dma，但是阻塞的dma已经不能发挥double buffer的好处了，显然，当访存计算比为1时，这个kernel是访存受限的。
3. 负载均衡不再使用round-robin算法，而是使每个从核的计算任务都是连续的。

优化前	上从核后	simd优化	double buffer	加速比
0.7亿	0.144亿	0.136	0.136	5.1

rho最后的优化结果

	优化前	优化后
64核	3.380	2.220
512核	27.350	17.630

H的优化

计算H的流程(C语言描述)

```
for(int i_my_batch=1;i_my_batch<n_my_batches_work;i_my_batch++)
{
    tab_atom_centered_coords();//21亿->4亿
    collect_batch_centers_p2();//0.17亿
    for(int i_index=0;i_index<batches_work_size(i_my_batch);i_index++)//21亿->34
亿
    {
        prune_radial_basis_p2();
        tab_local_geometry_p2();
        tab_trigonom_p0();//
        tab_wave_ylm_p0();
        evaluate_radial_functions_p0();
        evaluate_waves_p2();
        evaluate_xc_DFPT();
    }
    evaluate_first_order_H_polar_reduce_memory();
    //matrix_ddot();//8千8百万->一千五百万
    prune_density_matrix_sparse_from_dense();//15亿->1.2亿
}
```

消除wave数组的转置操作

```
!原始代码
real*8 :: wave(n_max_compute_ham,n_points)
real*8 :: contract(n_points,n_compute_c),wave_t(n_points,n_max_compute_ham)

wave_t=transpose(wave)
```

```
do i_compute=1,n_compute_c
    do i_point=1,n_points

contract(i_points,i_compute)=partition_tab(i_point)*wave_t(i_point,i_compute)*&
                                    (-
grid_coord(i_point)+v_hartree_gradient(i_point)&

 +dvxc_drho(i_point)*first_order_rho(i_point))
    end do
end do
call
dgemm("T","N",n_compute_c,n_compute_c,n_points,1.d0,contract,n_points,wave_t,n_p
oints,0.d0,&
            first_order_H_dense,n_compute_c)
```

上面是evaluate_first_order_H_polar_reduce_memory的部分原始代码，原始代码对wave这个数组做了一个转置，可以猜测这个转置的目的为了使二重循环对wave数组的访问的局部性更好，但是矩阵转置的操作本身很花时间，链接了xmath之后矩阵转置的时间比矩阵乘法的时间都长。所以把对wave矩阵的转置消除了，为了依然保证二重循环的局部性，可以把二重循环的循环顺序改变，同时改变contract数组的定义，修改后的代码如下所示：

```
real*8 :: wave(n_max_compute_ham,n_points)
real*8 :: contract(n_compute_c,n_points)

do i_point=1,n_points
    do i_compute=1,n_compute_c

contract(i_compute,i_point)=partition_tab(i_point)*wave(i_compute,i_point)*&
                                    (-
grid_coord(i_point)+v_hartree_gradient(i_point)&

 +dvxc_drho(i_point)*first_order_rho(i_point))
    end do
end do
call dgemm("N","T",n_compute_c,n_compute_c,n_points,1.d0,contract,n_compute_c,&
            wave,n_max_compute_ham,0.d0,first_order_H_dense,n_compute_c)
```

优化前	优化后	加速比
6.5亿	3.2亿	2

matrix_ddot优化

这个kernel便是上一节中的二重循环，有了rho的经验之后，这个kernel既没有用simd也没有用double buffer。

优化前	优化后	加速比
0.89亿	0.16亿	5.6

prune_density_matrxi_sparse_from_dense优化

优化前	优化后	加速比
15亿	1.2亿	12.5

H最后的优化结果

	优化前	优化后
64核	4.86	3.15
512核	35.28	21.10

大kernel

rho

tab_atom_centered_coords

这个函数运行了40亿个时钟周期，占比66%，优化后为4亿个时钟周期。

prune_density_matrix_sparse_polar_reduce_memory

这个函数运行时间为4亿个时钟周期，优化后为3亿个时钟周期，优化的方法是对称矩阵可以减少一半访存量，同时离散访存可以通过预取把gld操作转换为更快的dma操作，充分利用带宽。

H

	大kernel	小kernel
rho	2.56	2.22
H	14.28	4.89

热点分析

tab_atom_centered_coords	**10%**
prune_radial_basis	4%
tab_local_geometry	
tab_trigonom	
tab_wave_ylm	
evaluate_radial_function	
evaluate_waves	
pz_lda	
dgemm+ddot	
pre_reduction	
reduction	82%

一些失败的尝试

对i_index_loop的优化

在rho的优化这一节中可以看到这个循环占比31%，这个循环的内部是七个小的函数，一开始的思路是整个循环都上从核运行，但是最后的结果是运行时间从21亿个时钟周期减速为34亿个时钟周期，分析原因为这个循环所涉及的数组太多了，而且有不少离散访存，所以即使开了cache也没啥用。后来转变思路，把这个循环内部的小函数单独上从核，选取了tab_local_geometry_p2和tab_trigonom_p0。tab_local_geometry_p2这个函数的特点是需要访存的数组为4个，有三个为连续访存，一个为离散访存，而且计算量比较大，上从核运行后运行时间从2亿个时钟周期减速为8亿个时钟周期。tab_trigonom_p0这个函数的特点是需要访存的数组只有两个，而且都是连续访存，也有一定的计算量，但是上从核运行之后运行时间从1.9亿个时钟周期减速为6.5亿个时钟周期。分析原因发现有两个，其一是反复上从核有不小的开销，其二是这两个kernel的任务量太小，不够64个从核分。总而言之，这个循环整个上从核会因为离散访存太多而减速，里面的函数单独上从核又会因为任务量太小而不能抵消反复上从核的开销，还有一个思路是循环分裂，但是这样又太浪费空间，所以这个循环并不能被优化。

主从协同

i_index_loop和prune_density_matrix_sparse_polar_reduce_memory之间是没有数据依赖的，由上面可知，i_index_loop是没法上从核的，所以prune_density_matrix_sparse_polar_reduce_memory在从核上运行时主核上可以同时运行i_index_loop从而隐藏prune_density_matrix_sparse_polar_reduce_memory的运行时间，然而由于xmath的存在，i_index_loop的某些内置函数是调用了从核的，比如log函数，所以这次尝试也失败了。

既然prune_density_matrix_sparse_polar_reduce_memory不能和i_index_loop主从协同，但是可以和tab_atom_centered_coords主从协同，但是结果显而易见，tab_atom_centered_coords上从核后加速比是远大于2的，所以主从协同不如两个函数都上从核。

总结

神威海洋之光是严重主核和从核之间带宽受限的，所以除非是特别密集的计算，都不能很好的发挥神威海洋之光从核的计算能力，对于不是特别计算密集的kernel来说，都可以视为访存受限的，所以说即使这个kernel全是访存，进行优化也不是没有意义的。计算密集型的kernel从核加速效果最好，仅有连续访存或近似连续访存的kernel从核加速效果一般，加速比在2-12之间，有大量离散访存的kernel利用从核反而有减速效果。