Out-of-Core Cholesky Factorization Algorithm on GPU and the Intel MIC Co-processors

Ben Chan (Chinese University of Hong Kong)
Nina Qian (Chinese University of Hong Kong)
Mentors: Ed D'Azevedo (ORNL)
Shiquan Su (UTK)
Kwai Wong (UTK)
5th Augest 2013

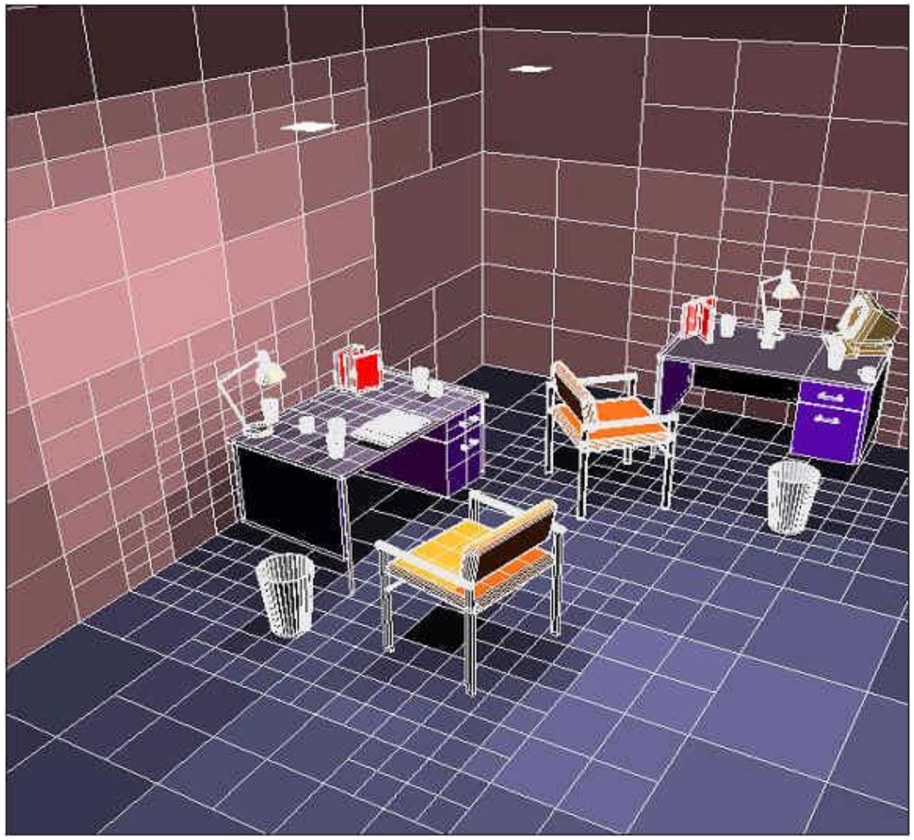
Outline

- Motivation: Large scale radiosity problem
  - Introduction to view3d program
  - Connection with out-of-core algorithm
  - Performance on Keeneland (GPU) and Beacon (MIC)
- Factorization Algorithm
  - Theory
  - Performance on Keeneland (GPU)
  - Using MIC

View3D for large scale radiosity problem

- By Stepthen-Boltzmann's equation, radiation reflects the objects temperature.
- View factor measures the radiation which leaves one surface and strikes another surface.

View3D program: Parallel calculation of the view factor between any two surfaces and generate the view factor matrix F.



https://www.cs.duke.edu/courses/cps124/spring04/notes/08_rendering/

Connection to out-of-core algorithm

- Stepthen-Boltzmann's equation: $GR_i = \sigma T_i^4$,
  where $G = \delta_{ij} - \phi_i A_i F_{ij}$.

  Transformed radiosity matrix G: (SPD)

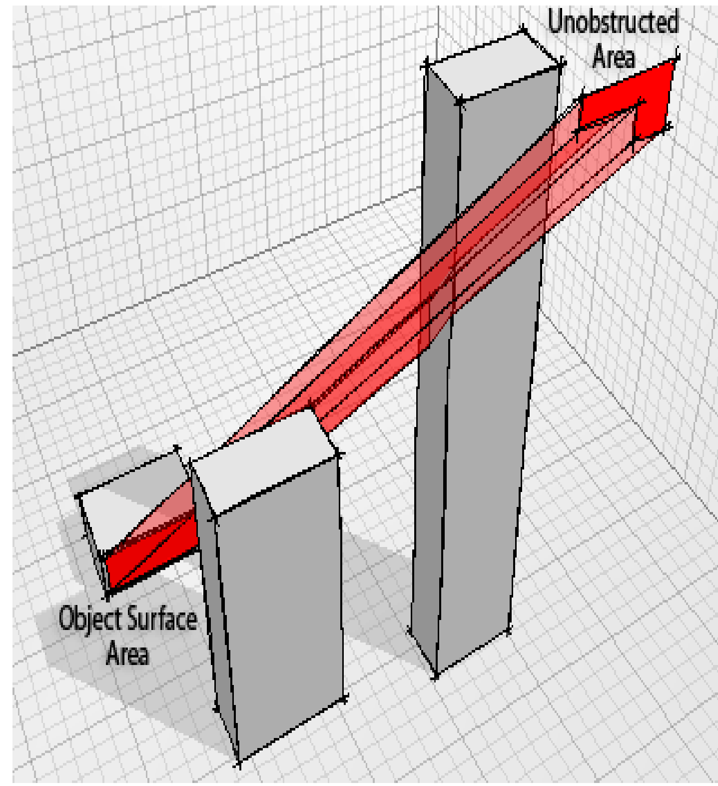
$$G = \begin{pmatrix} \frac{A_1}{\phi_1} - A_1F_{11} & -A_2F_{12} & \cdots & -A_NF_{1N} \\ -A_1F_{21} & \frac{A_2}{\phi_2} - A_2F_{22} & \cdots & -A_NF_{2N} \\ \vdots & & & \vdots \\ -A_1F_{N1} & -A_2F_{N2} & \cdots & \frac{A_N}{\phi_N} - A_NF_{NN} \end{pmatrix}$$

- Radiosity problem ➔ solve system of linear equation ➔matrix factorization ➔ out-of-core algorithm

View3D on Keeneland (GPU) and Beacon (MIC)

Various of pairs of surfaces

- With potential obstruction → Calculate inside CPU
- Without potential obstruction → GPU (Keeneland) / MIC (Beacon)



http://naturalfrequency.com/articles/shadingcalculations

Implementation of View3D on MIC

- Beacon: each node has 16 processors and 4 MIC cards
  - Assign one MIC card to each core
  - Use offload with shared VM

Data in shared virtual memory:

```
Surface_MIC * _Cilk_shared DEV_MIC_srf;
double * _Cilk_shared DEV_ans;

DEV_MIC_srf=(_Cilk_shared Surface_MIC *)
_Offload_shared_malloc(sizeof(Surface_MIC)*(vfCtrl.nAllSrf+1));

DEV_ans=(_Cilk_shared double *)
_Offload_shared_malloc(sizeof(double)*np*nq);
```

```
int num_devices;
#ifdef __INTEL_OFFLOAD
num_devices = _Offload_number_of_devices();
#else
num_devices = 0;
#endif

if (num_devices == 0)
{
        HOST_Comp(DEV_MIC_srf,DEV_ans,rank,np , nq ,npr
ow, npcol, myrow, mycol, nb);
}

if (num_devices!=0)
{
        _Cilk_spawn _Cilk_offload_to(rank%num_devices)
MIC_Comp(rank,np , nq ,nprow, npcol, myrow, mycol, nb);

        View3D( srf, base, possibleObstr, A, &vfCtrl ,n
p,nq,nprow,npcol,myrow,mycol,nb,Coef );

_Cilk_sync;}
```

Unobstructed part:
Offload to MIC

Obstructed part:
Do in Host

Synchronize DEV_ans

Performance on Keeneland (GPU) and Beacon (MIC)

- Case comparison:
  - L shape case (no obstruction)
  - Total number of surfaces: 20000
  - Processor grid: 6 x 6, NB = 64

Determine possible obstruction		Calculation of unobstructed cases	
Keeneland	Beacon	Keeneland	Beacon
1.795 sec	2.149 sec	6.507 sec	111.09 sec

- Future directions for view3d based on MIC:
  - Enhance stability
  - Multiple MIC cards
  - Directive offload

Outline

- Motivation: Large scale radiosity problem
  - Introduction to view3d program
  - Connection with out-of-core algorithm
  - Performance on Keeneland (GPU) and Beacon (MIC)
- **Factorization Algorithm**
  - Theory
  - Performance on Keeneland (GPU)
  - Using MIC

Cholesky Factorization

- Factorize any symmetric positive-definite (SPD) matrix into the form $L \times L^t$

$$\begin{pmatrix} 4 & 8 & 2 \\ 8 & 17 & 3 \\ 2 & 3 & 11 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 0 & 0 \\ 4 & 1 & 0 \\ 1 & -1 & 3 \end{pmatrix} \begin{pmatrix} 2 & 4 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 3 \end{pmatrix}$$

- Rewrite $Ax = b$ into $\begin{cases} Ly = b \\ L^t x = y \end{cases}$

How?

Suppose such factorization exists:

Consider a block matrix form of A and L

$$A = \begin{pmatrix} A_{11} & (A_{21})^t \\ A_{21} & A_{22} \end{pmatrix}; L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}$$

From $A = L \times L^t$, we have

$$L_{11} = chol(A_{11})$$

$$L_{21} = A_{21}\left((L_{11})^t\right)^{-1}$$

$$L_{22} = chol(A_{22} - L_{21}(L_{21})^t)$$

Right-looking method and Left-looking method

OOC Approach of the Factorization

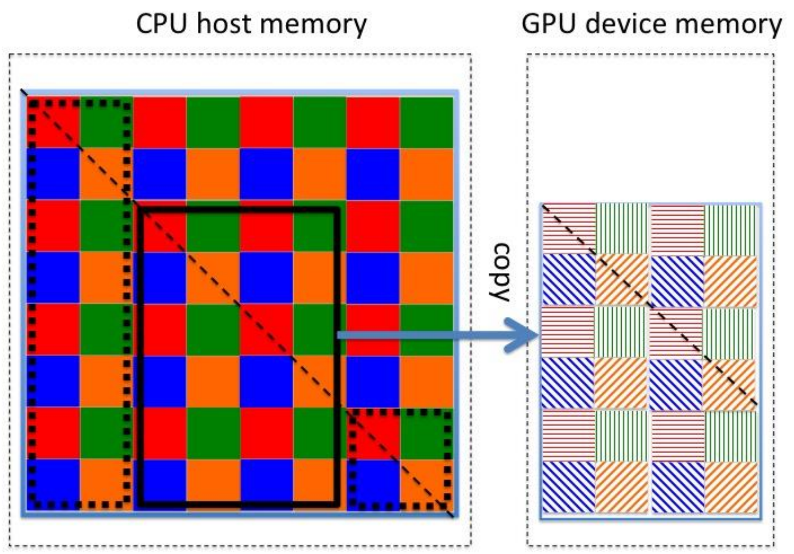
- Hardware accelerators in parallel computers
  - GPU in Kraken and Keeneland
    MIC in Beacon
  - Computing “core” of the algorithm (or “device”)
    Data stored “Out-of-Core” (the “host”)
- Combine two standard methods together

  Right-looking method
  Left-looking method

OOC Approach of the Factorization

- Use a 2D-block cyclic distribution; column-major storage

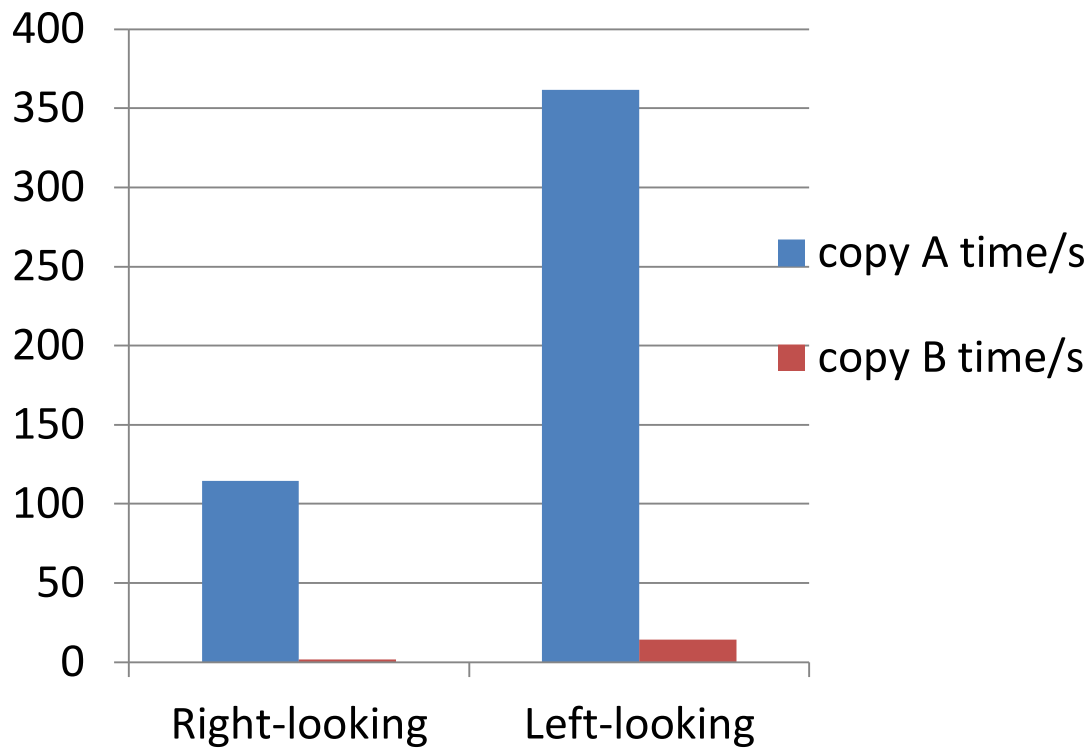
- Chop the matrix into pane
- Copy a panel into core
  ➔ left-looking method
  ➔ right-looking method
- Continue to next panel



Host-to-Host Data Transfer

	Right-looking	Left-looking
copy A data (TB)	25.4	96.5
copy B data (TB)	2.5	22.9

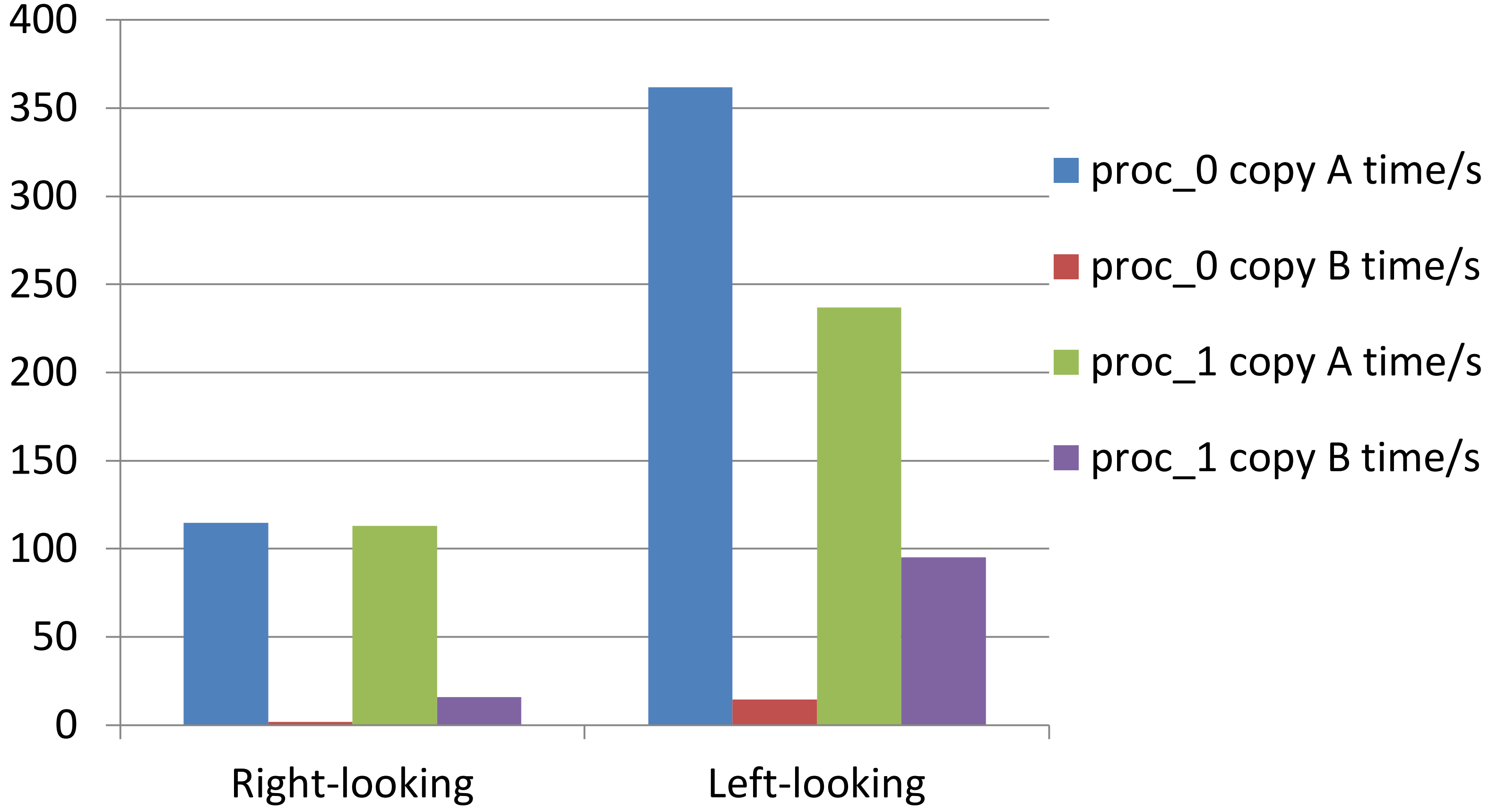
- Tested on Keeneland
- Matrix size 518400
- Block size 64
- Processor grid 27x27
- Chop 12 panels



Timing results are affect by the workload of different processes!

Host-to-Host Data Transfer

	Right-looking	Left-looking
copy A data (TB)	25.4	96.5
copy B data (TB)	2.5	22.9

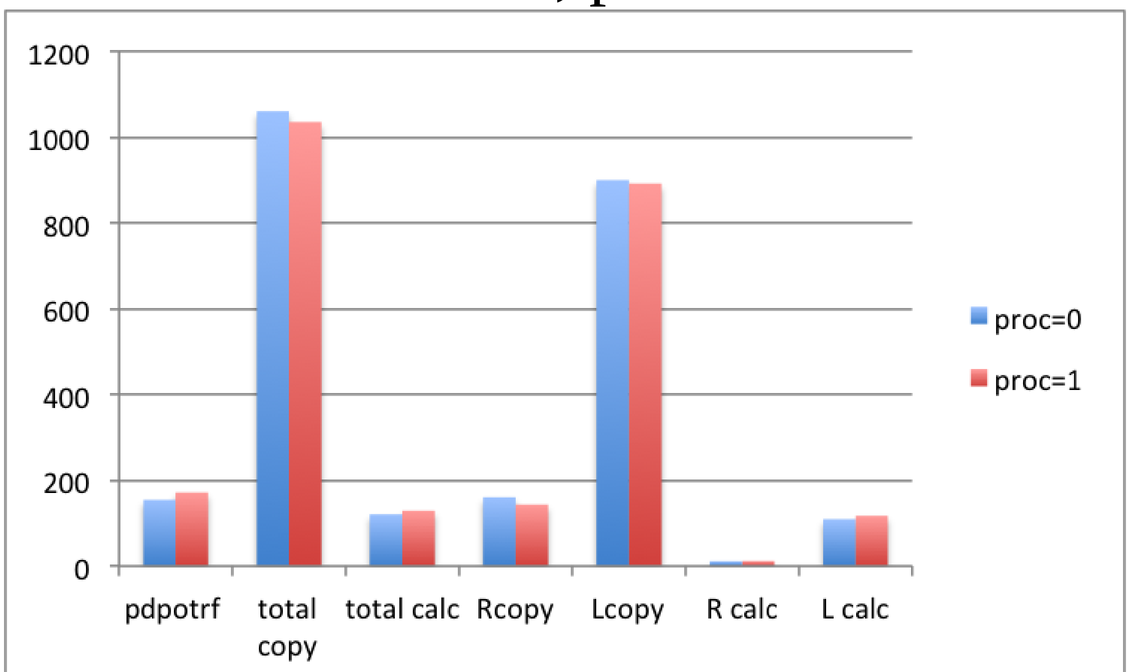


Performance on Keeneland

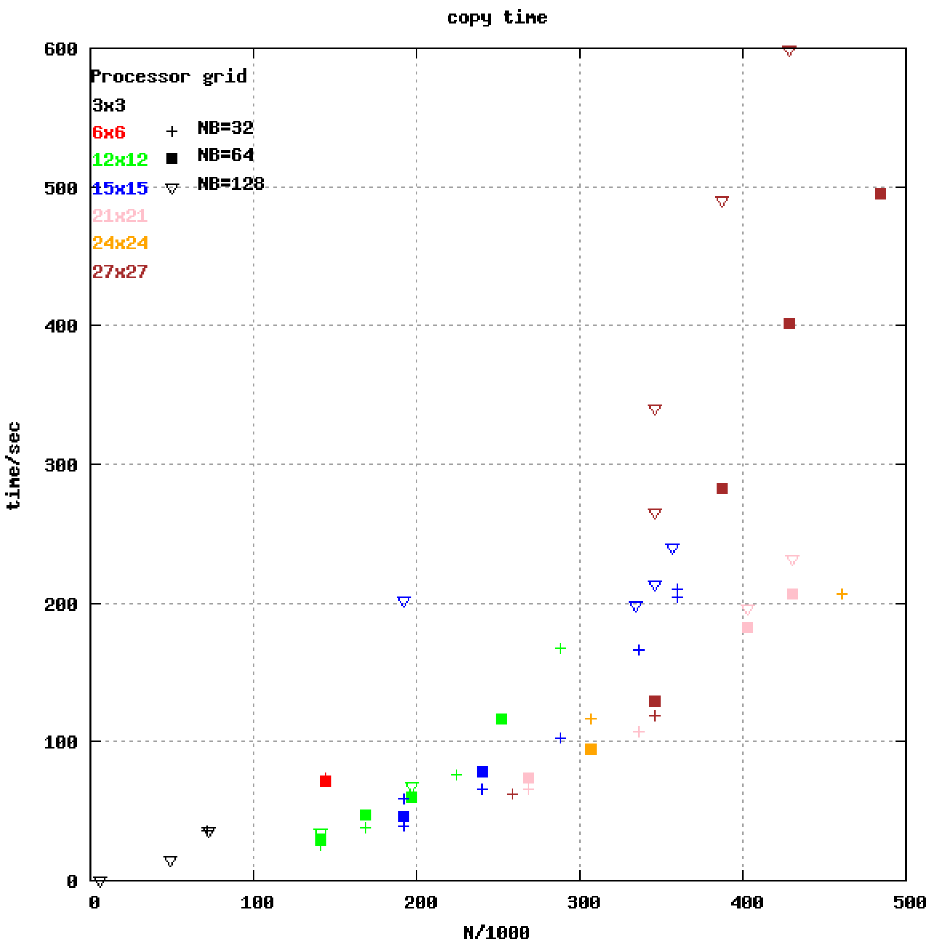
- Tested cases
  - Total cases: 117
    - Successful: 65
  - Matrix size N from 49152 to 552960
  - NB = 32, 64, 128
  - Processor grid: 3 x 3, 6 x 6, 12 x 12, 15 x 15, 21 x 21, 24 x 24, 27 x 27
  - Most cases fix 2-panels

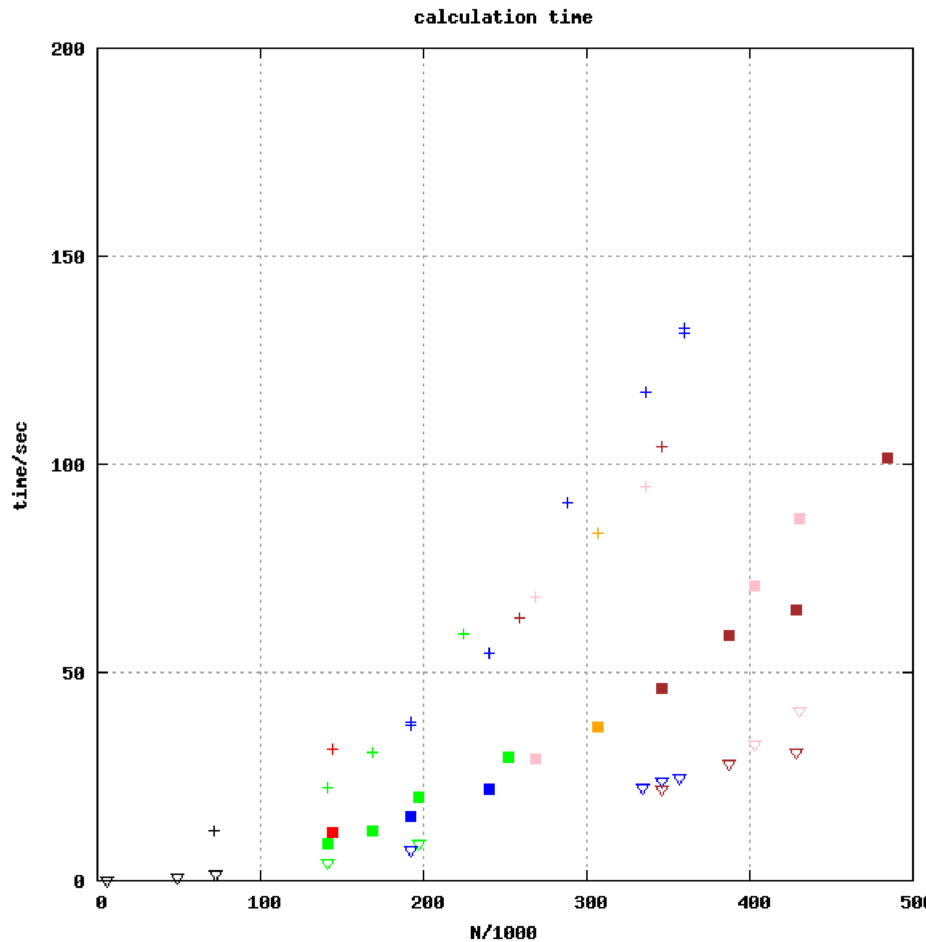
Performance on Keeneland

- Biggest successful case:
  - Matrix size: N=552960 (73% of maximum size)
  - Processor grid: 27x27, NB=64
  - Divided into 12 panels
- Total time: 1366 secs, performance: 56 GFLOPS/C

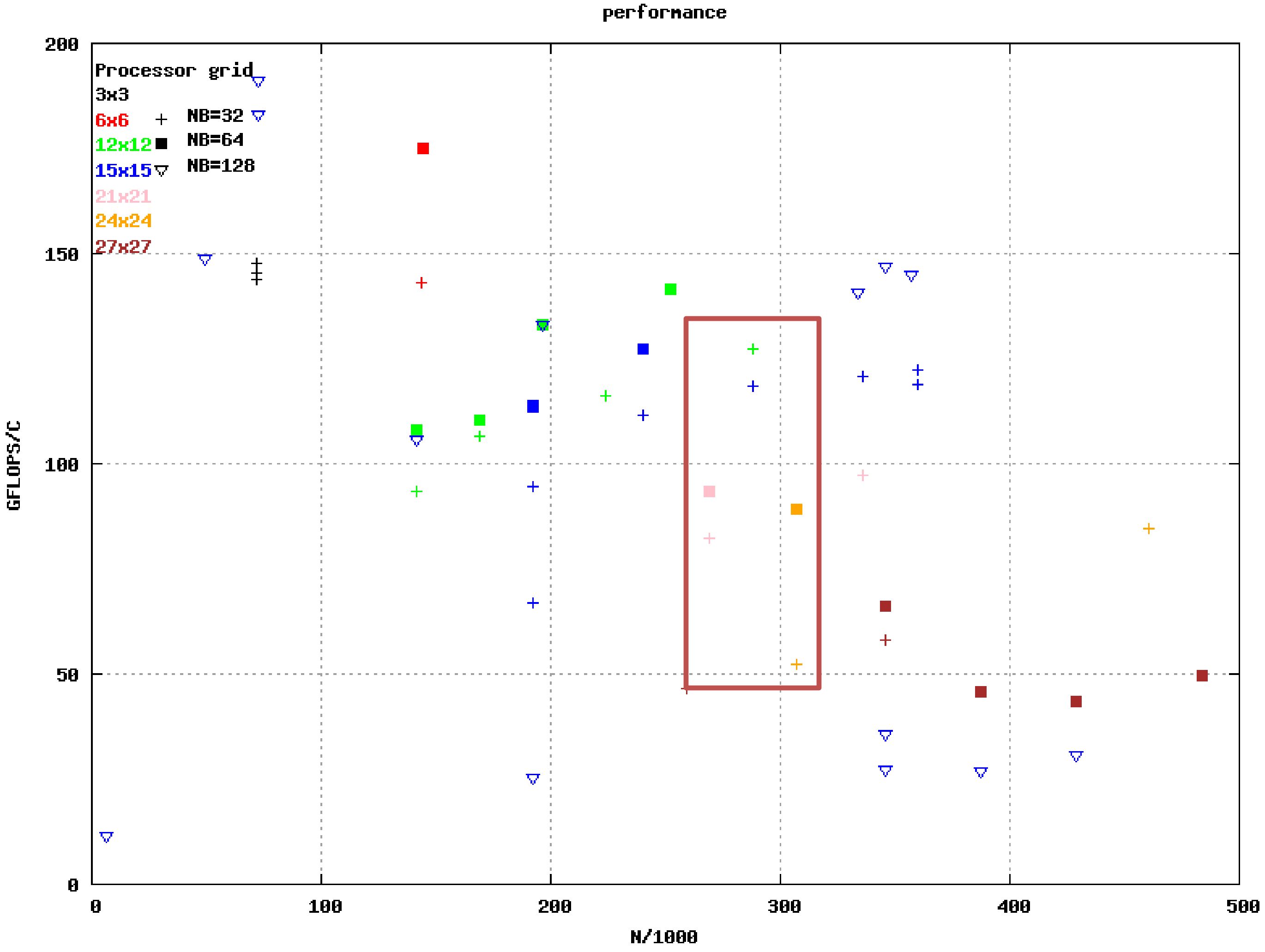


- Observations:
  - set NB = 128 for small case
    - Better performance
      (calculation > communication)
  - set NB = 64 for big case
    - More stable N > 400000
    - Better performance
      (communication > calculation)





- Observations:
  - Fixed matrix size, smaller processor grid has higher performance (less host-to-host data transfer)



Outline

- Motivation: Large scale radiosity problem
  - Introduction to view3d program
  - Connection with out-of-core algorithm
  - Performance on Keeneland (GPU) and Beacon (MIC)
- **Factorization Algorithm**
  - Theory
  - Performance on Keeneland (GPU)
  - **Using MIC**

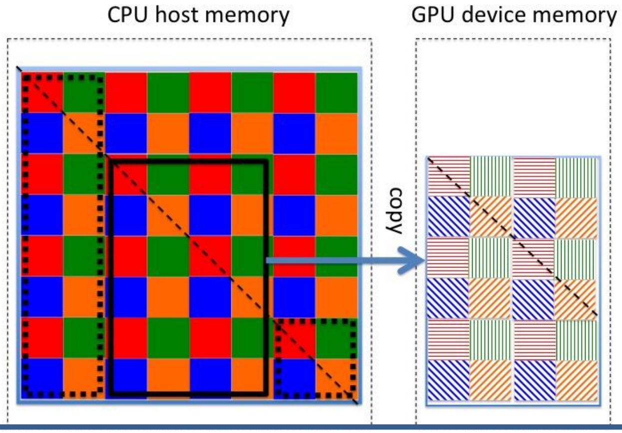
Coding for MIC

- C/C++ codes for the algorithm

  offload to GPU ➔ offload to MIC

  – Allocate/free memory
  – Host-device data transfer
  – CUBLAS function calls
- Compiling and Linking

Allocate/free device memory



- Use pragma offload

  alloc_if() free_if() modifiers

```
double *Y = (double*) malloc(n*sizeof(double));
#pragma offload_transfer target(mic:MYDEVICE) nocopy(Y:length(n) alloc_if(1) free_if(0))
#pragma offload_transfer target(mic:MYDEVICE) nocopy(Y:length(n) alloc_if(0) free_if(1))
```

```
int *p = (int*) malloc(1*sizeof(int));
int *q = (int*) malloc(1*sizeof(int));
#pragma offload_transfer target(mic) nocopy(p:length(1000) alloc_if(1) free_if(0))
#pragma offload_transfer target(mic) nocopy(q:length(1000) alloc_if(1) free_if(0))
```

```
offload error: address range partially overlaps with existing allocation
```

- Problems
  - length() and alloc_if() creates a mapping between host memory and device memory within a certain interval of addresses
  - the pragma offload directives are not designed to support what we want!

Allocate/free device memory

```
intptr_t offload_Alloc(size_t size){
    intptr_t ptr;
    #pragma offload target(mic:MYDEVICE) out(ptr)
    {
        ptr = (intptr_t) memalign(64, size);
    }
    return ptr;
}

void offload_Free(void* p){
    intptr_t ptr = (intptr_t)p;
    #pragma offload target(mic:MYDEVICE) in(ptr)
    {
        free((void*)ptr);
    }
}
```

```
#ifdef USE_MIC
    dY = (double*) offload_Alloc(isizeY*elemSize);
#else
    cublasAlloc( isizeY, elemSize, (void **) &dY );
#endif
```

```
if (dAtmp != 0) {
    #ifdef USE_MIC
        offload_Free(dAtmp);
    #else
        CUBLAS_FREE( dAtmp );
    #endif
    dAtmp = 0;
};
```

Host-device data transfer

- Use pragma offload again!

```
#pragma offload_transfer target(mic) in(Y[1:n]: alloc_if(0) free_if(0) into(dY[1:n]))
```

```
offload error: cannot find data associated with pointer variable 0x214d4c0
```

- Use a buffer

  Allocate buffer memory on host

  Allocate buffer memory on device with alloc_if()

- 1- Copy Y into buffer on host
- 2- Offload transfer buffer to device
- 3- Copy buffer on device into dY

CUBLAS function calls

```
CUBLAS_DGEMM(
        CUBLAS_OP_N,CUBLAS_OP_N, mm,nn,kk,
        zalpha,  (double *) dA(lrA1,lcA1), ldAtmp,
                 (double *) dB(lrB1,lcB1), ldBtmp,
        zbeta,   (double *) dC(lrC1,lcC1), ldC );
```

```
offload_dgemm("N", "N", &mm, &nn, &kk,
        &zalpha,  (double *) dA(lrA1,lcA1), &ldAtmp,
                  (double *) dB(lrB1,lcB1), &ldBtmp,
        &zbeta,   (double *) dC(lrC1,lcC1), &ldC );
```

```
void offload_dgemm(const char *transa, const char *transb, const MKL_INT *m, const MKL_INT *n, const MKL_INT *k,
                   const double *alpha, const double *a, const MKL_INT *lda, const double *b, const MKL_INT *ldb,
                   const double *beta, double *c, const MKL_INT *ldc){
/*
 * perform dgemm on the device. a,b,c pre-exist on the device
 */
    intptr_t aptr = (intptr_t)a;
    intptr_t bptr = (intptr_t)b;
    intptr_t cptr = (intptr_t)c;
    #pragma offload target(mic:MYDEVICE) in(transa,transb,m,n,k:length(1)) \
                                         in(alpha,lda,ldb,beta,ldc:length(1))
    {
        dgemm(transa,transb,m,n,k,alpha,(double*)aptr,lda,(double*)bptr,ldb,beta,(double*)cptr,ldc);
    }
}
```

Compilation

Compilation:

mpiicc -c ooc_offload.cpp

↓

ooc_offload.o, ooc_offloadMIC.o

Linking:

```
mpiicc -o pdlltdriver2.exe\
        main.cpp lib.a ooc_offload.o \
        -llibraries
```

↓

pdlltdriver2.exe

Code tested on Beacon

- Use 4 MICs per node, 64 nodes
- Matrix size 368640
- Block size 512
- Processor grid 12x12
- Chop two panels
  - ➔ 47.10 GFLOPS per process
  - ➔ less than 1/3 of the speed with GPU!

Future Work

- MIC
  - Asynchronous offload
  - More optimization
- Algorithm
  - More parallelism ?
- Performance evaluation

Out-of-Core Cholesky Factorization Algorithm on GPU and the Intel MIC Co-processors

Ben Chan (Chinese University of Hong Kong)
Nina Qian (Chinese University of Hong Kong)
Mentors: Ed D'Azevedo (ORNL)
Shiquan Su (UTK)
Kwai Wong (UTK)
5th Augest 2013