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

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

## OVERVIEW

➢A hybrid method for the Cholesky factorization of large dense SPD matrices.

➢Combine two standard procedures together: right-looking method and left-looking method.

➢The problem matrix is stored in host CPU memory ( out-of-core )
- ❑High amount of data storage

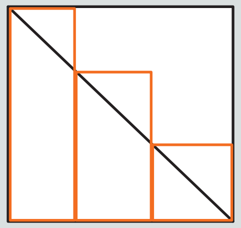
➢Offload heavy computational parts to devices ( in-core )
- ❑High computational power
- ❑GPU and MIC

## OOC CHOLESKY FACTORIZATION ALGORITHM

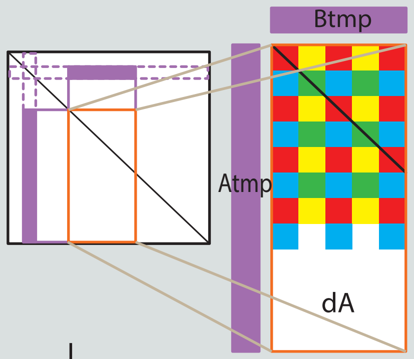
There is an existing implementation of the algorithm in double precision using GPU as core devices. **ScaLAPACK** is used for calculation within host CPU; **CUBLAS** is used for calculation within the device GPU.

Data distribution
- ➢2D-block-cyclic distribution on an adjustable rectangular process grid
- ➢Stored in column-major order

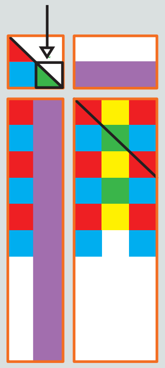
### General Procedure:



➢Chop the matrix into panels
- ❑The panel is transferred to core
- ❑Size of panel is limited to core memory
- ❑Different ways of determining each panel size can fine-tune the performance




For each panel, compute left-looking updates:
- ➢With every factorized block-columns on the left, update *dA <- dA - Atmp x Btmp* by subroutine `Cpdsyrk_hhd`
- ➢Involve host-host-device data transfer using **BLACS** and **CUBLAS**
- ➢Compute by calling `dsyrk` (diagonal part) and `dgemm` (off-diagonal part) of **CUBLAS**



Followed by right-looking updates:
For every block along the diagonal,
- ➢call `pdpotrf` to obtain its lower triangular factor;
- ➢call `pdtrsm` to update the submatrix under the block;
- ➢call `Cpdsyrk_hhd` to update the trailing matrix with the updated column

## 15x15 PROCESS GRID CASE

Machine: **Keeneland**
Test case:
- ➢Problem size = 360000
- ➢Block size = 32
- ➢Process grid = 15x15
- ➢Panel division = two equal-width panels

### Timing results

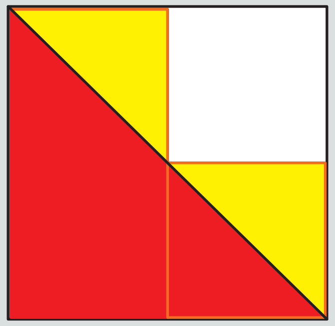
| time /second | Procedure | number of calls |
|---|---|---|
| 1.4 | Copy panel host-device | 2 |
| 241.1 | Right looking column factorization | 11250 |
| 11.2 | Host-device data transfer | |
| 229.8 | Computation of `pdpotrf`, `pdtrsm` | |
| 159.3 | Right-looking update `Cpdsyrk_hhd` | 11248 |
| 74.7 | Host-host data transfer | |
| 16.4 | host-device data transfer | |
| 67.5 | Computation of `dsyrk`, `dgemm` | |
| 177.4 | Left-looking update `Cpdsyrk_hhd` | 5625 |
| 88.0 | Host-host data transfer | |
| 25.1 | host-device data transfer | |
| 64.0 | Computation of `dsyrk`, `dgemm` | |

Total time = **579.9** seconds
Overall performance = **118.8** GFLOPS/PROC
( vs GPU peak performance 665 GFLOPS )

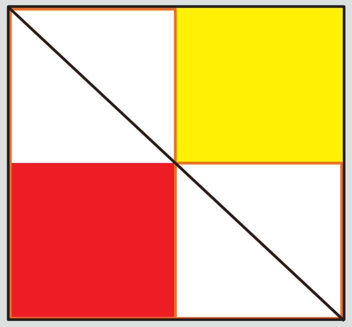
Host-host data transfer is shown to be a dominating cost of time. Its analysis is helpful for understanding the behavior of the algorithm when we further scale up the problem size.

## QUANTIFYING HOST-HOST DATA TRANSFER AMOUNT

Problem size, grid size and panel division policy are factors determining the amount of host-host data transfer. For a typical case with two equal-width panels, we can consider the transfer in right-looking part and left-looking part separately.



Left-looking part



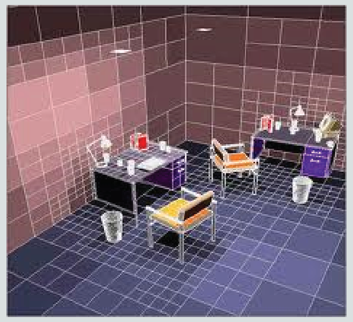
Right-looking part

*Red area x Q and yellow area x P* gives an upper bound of the data transfer amount, where Q is the number of columns, P is the number of rows in the processor grid.

## APPLICATION: LARGE SCALE RADIOSITY PROBLEM

### Background

➢Thermal radiosity problem exists in thermal engineering and other fields. It helps us to obtain temperature information or generate realistic diffuse reflection. View factor measures the radiation which leaves surface 1 and strikes surface 2.



Radiosity problem in computer science, objects in space divided into subsurface to calculate view factor.

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

### Brief Algorithm

➢For each pair of surfaces that are facing each other generate a potential obstruction list by several excluding tests which eliminate the number of obstructing surfaces.

➢If the potential obstruction list is empty after tests, then use an appropriate view factor formulation to calculate for the unobstructed pairs. If not, then project shadow to the surface of object surface to do further obstruction test and calculation.

➢GPU-based parallel version of View3D decomposes the matrix to each processor and command CPU to calculate the obstructed pairs. GPU is supposed to calculate all the view factors and pass the unobstructed data back to CPU.

### View Factor Matrix to SPD Matrix

➢By Stepthen-Boltzmann's equation, $GR_i = \sigma T_i^4$, where $G = \delta_{ij} - \phi_i A_i F_{ij}$. Symmetry of matrix G is assured in view factor formulations $(A_i F_{ji} = A_j F_{ij})$, also the diagonal dominates other entries. Hence the transformed matrix G of view factor matrix F is an SPD matrix which can be solved using OOC algorithm.

Transformed radiosity matrix G

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

## CONTACT INFO

| Ed D'Azevedo | Shiquan Su | Kwai Wong |
|---|---|---|
| dazevedoef@ornl.gov | ssu2@utk.edu | kwong@utk.edu |