学際大規模情報基盤共同利用・共同研究拠点 公募型共同研究 平成25年度採択課題
jh130015-NA10
5th Symposium
Joint Usage / Research Center for Interdisciplinary Large-scale Information Infrastructures

課題代表者氏名（所属）藤澤克樹（中央大学 & JST CREST）

研究課題名
内点法アルゴリズムの並列計算による超大規模半正定値計画問題の解決

Peta-scale General Solver for Semidefinite Programming Problems with over Two Million Constraints

High-Performance General Solver for Extremely Large-scale Semidefinite Programming Problems (SDPs)

1. Mathematical Programming : one of the most central problems in mathematical optimization.
2. Many Applications : combinatorial optimization, control theory, structural optimization, quantum chemistry, sensor network location, data mining, etc.

- **SDPARA** is a parallel implementation of the interior-point method for Semidefinite Programming (SDP)
- Parallel computation for **two major bottleneck parts**
  - **ELEMENTS** ⇒ Computation of Schur complement matrix (SCM)
  - **CHOLESKY** ⇒ Cholesky factorization of Schur complement matrix (SCM)
- **SDPARA** could attain high scalability using **16,320 CPU cores** on the TSUBAME 2.0 supercomputer and some techniques of processor affinity and memory interleaving when the generation of SCM constituted a bottleneck.
- **With 4,080 NVIDIA M2050 GPUs** on the TSUBAME 2.0 supercomputer, our implementation achieved **1.018 PFlops** in double precision for a large-scale problem with over two million constraints.

SemiDefinite Programming (SDP)

The number of constraints of the primal problem

Primal

$$\left.\begin{array}{ll}\text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{A}_p \bullet \boldsymbol{X} = b_p \ (p = 1, 2, \ldots, m), \ \boldsymbol{X} \in \mathcal{S}^n_+ \end{array}\right\} \quad (1)$$

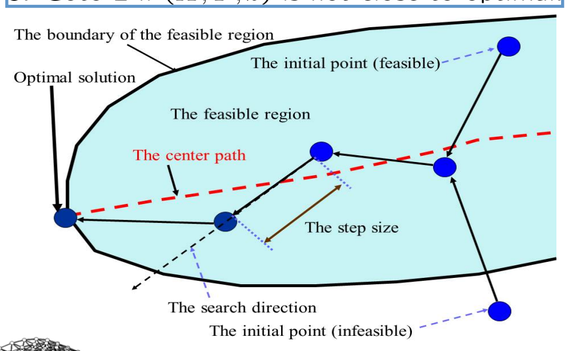
Dual

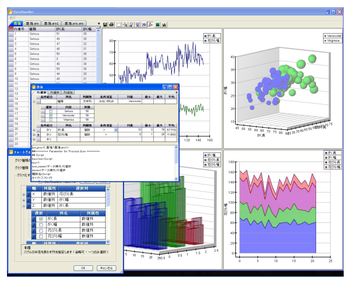
$$\left.\begin{array}{ll}\text{maximize} & \sum_{p=1}^{m} b_p z_p \\ \text{subject to} & \sum_{p=1}^{m} \boldsymbol{A}_p z_p + \boldsymbol{Y} = \boldsymbol{A}_0, \ \boldsymbol{Y} \in \mathcal{S}^n_+ \end{array}\right\}. \quad (2)$$

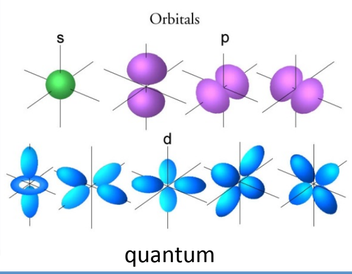
The matrix size

$\mathcal{S}^n$: $n \times n$ symmetric matrices
$\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} Y_{ij}$
$\boldsymbol{X} \in \mathcal{S}^n_+$ ($\mathcal{S}^n_{++}$) : $\boldsymbol{X} \in \mathcal{S}^n$ is positive semidefinite (or positive definite)
$\boldsymbol{A}_p \in \mathcal{S}^n$ $(p = 0, 1, \ldots, m)$ and $\boldsymbol{b} \in \mathbb{R}^m$

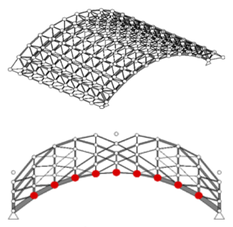
1. Choose an initial point $(\boldsymbol{X}, \boldsymbol{Y}, z)$ with $\boldsymbol{X} \succ \boldsymbol{O}, \boldsymbol{Y} \succ \boldsymbol{O}$.
2. Compute Search Direction $(d\boldsymbol{X}, d\boldsymbol{Y}, dz)$.
3. Compute Step Length α_p, α_d. $\boldsymbol{X} + \alpha_p d\boldsymbol{X} \succ \boldsymbol{O}, \boldsymbol{Y} + \alpha_d d\boldsymbol{Y} \succ \boldsymbol{O}$
4. Update $(\boldsymbol{X}, \boldsymbol{Y}, z) \leftarrow (\boldsymbol{X} + \alpha_p d\boldsymbol{X}, \boldsymbol{Y} + \alpha_d d\boldsymbol{Y}, z + \alpha_d dz)$.
5. Goto 2 if $(\boldsymbol{X}, \boldsymbol{Y}, z)$ is not close to optimal.

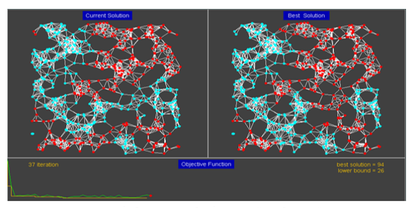






quantum


torus

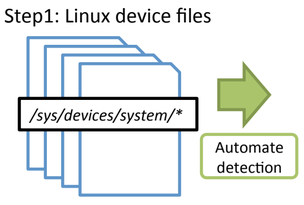

combinatorial optimization

The major bottleneck parts of PDIPM for the various types of SDP problems

1. *CHOLESKY* : Sparse SCM
   e.g.) the sensor network location problem and the polynomial optimization problem
   Parallel Sparse Cholesky factorization of sparse SCM
2. ***ELEMENTS*** : m < n (not m >> n), and Fully Dense SCM
   e.g.) the quantum chemistry problem and the truss topology problem
   Efficient Hybrid (MPI-OpenMP) parallel computation
   Automatic configuration for CPU Affinity and memory interleaving
3. ***CHOLESKY*** : m >> n, and Fully Dense SCM
   Massively parallel dense Cholesky factorization using GPUs
   e.g.) the combinatorial optimization problem the quadratic assignment problem (QAP)
   Overlapping techniques for computation, PCI-Express comm., and MPI comm.
   Our implementation achieved 1.018 PFlops in double precision for large-scale Cholesky factorization using 2,720 CPUs and 4,080 GPUs.

Automate configuration of CPU Affinity and memory allocation policy

Step1: Linux device files



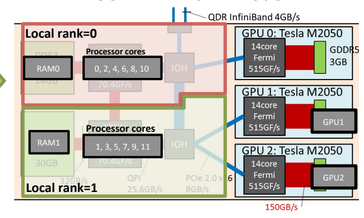
Step2: Processor mapping table

Processor ID	Package ID	Core ID
0	0	0
1	1	1
2	0	2
…		

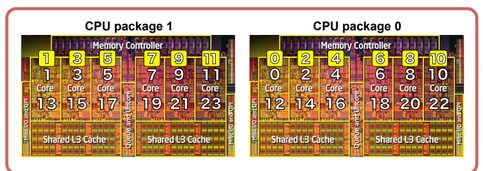
Step3: thread–Processor assignment table

Thread ID	Scatter	Compact
0	0	0
1	1	2
2	2	4
…		

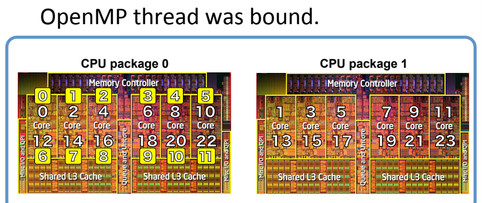
TSUBAME 2.0 HP node



Scatter-type Affinity
distributes OpenMP threads as evenly as possible across the entire system.



Compact-type Affinity
binds the (n+1)-th OpenMP thread in a free-thread context as close as possible to the thread context in which the n-th OpenMP thread was bound.



Parallel Computation for *ELEMENTS*

```
set_mempolicy(interleaving)
B = O
for l = 1, 2, ··· , h do
  for j ∈ {j | F_j^l ≠ O} parallel(MPI+thread) do
    set_affinity(scatter)
    for i ∈ {i | F_i^l ≠ O} do
      Selects F_1, F_2 or F_3 and compute B_ij^l
      B_ij = B_ij + B_ij^l
    end
    unset_affinity()
  end
end
```

```
packages ← ∅
for thread_id = 0, 1, ··· , p − 1 do
  packages ← packages ∪ { get_package_id(thread_id, scatter) }
end
set_mempolicy(packages, MPOL_INTERLEAVE)  // set
```

```
#omp parallel {
  thread_id ← omp_get_thread_num()
  pinned(thread_id, scatter)
  ...
  unpinned()
}
```

	F_i^l	F_i^l	formulation
$\mathcal{F}_1$	dense	dense	$B_{ij}^l = ((\boldsymbol{X}^{-1})^l \boldsymbol{F}_i^l \boldsymbol{Y}^l) \bullet \boldsymbol{F}_j^l$
$\mathcal{F}_2$	dense	sparse	$B_{ij}^l = \sum_{(\gamma,\epsilon)} \left(\sum_{\alpha=1}^{n} [(\boldsymbol{X}^{-1})^l]_{\gamma\alpha} [\boldsymbol{F}_i^l \boldsymbol{Y}^l]_{\alpha\epsilon} \right) [\boldsymbol{F}_j^l]_{\gamma\epsilon}$
$\mathcal{F}_2$	sparse	sparse	$B_{ij}^l = \sum_{(\gamma,\epsilon)} \left(\sum_{(\alpha,\beta)} [(\boldsymbol{X}^{-1})^l]_{\gamma\alpha} [\boldsymbol{F}_i^l]_{\alpha\beta} \boldsymbol{Y}_{\beta\epsilon}^l \right) [\boldsymbol{F}_j^l]_{\gamma\epsilon}$

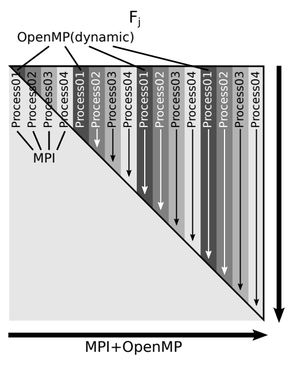


Fig. CPU time for Electronic4
- the memory interleaving is effective.
- the parameter set "*scatter* and *interleaving*" is fastest.

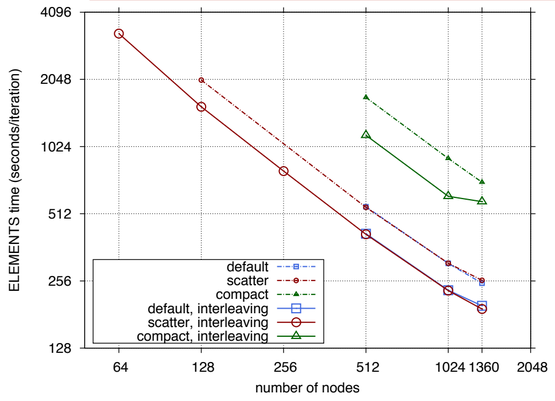
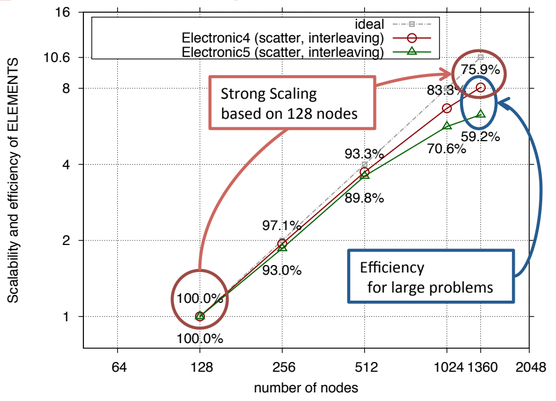


Fig. Scalability for Electronic4 and Electronic5
- high efficiency (75.9% for Electronic4)
- higher efficiency when solving an SDP problem larger than Electronic4 because the efficiency for Electronic4 is higher than that for Electronic5.



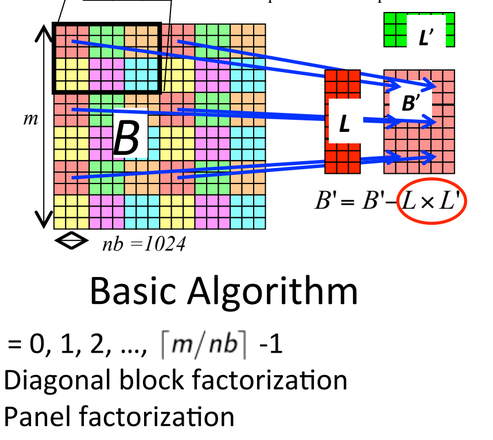
Parallel Computation for *CHOLESKY*

For problems with m >> n, high performance **CHOLESKY** is implemented for GPU supercomputers.
Key for petaflops is **overlapping computation, PCI-Express communication and MPI communication.**

Data Decomposition

- The dense matrix ***B(m x m)*** is distributed with 2D Block-Cyclic distribution with block size *nb*

Matrix distribution on 6 (=2x3) processes



Basic Algorithm

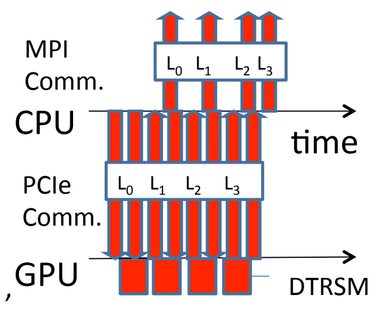
For k = 0, 1, 2, …, $\lceil m/nb \rceil$ -1
1. Diagonal block factorization
2. Panel factorization
   → **Compute L by GPU DTRSM**
3. Broadcast L, transpose L (L') and broadcast L'
4. **Update B' = B' – L x L' with fast GPU DGEMM**

Design Strategy

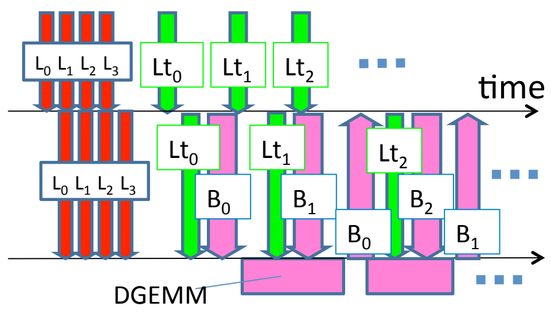
Our target is large problems with m > 2 million
→ GPU memory is too small. Matrix data are usually placed on host memory
- Blocksize nb should be sufficiently large to mitigate CPU-GPU PCIe communication
- We still suffer from heavy PCIe communication

→ nb=1024
→ Overlap GPU comp., PCIe comm., and MPI comm. in Step 2, 3 and 4

Producer of panel L

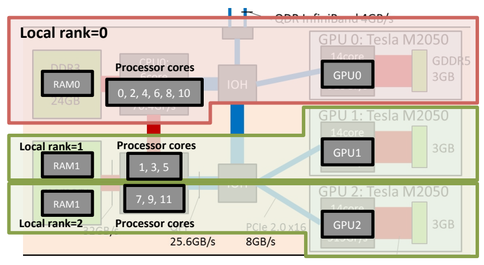


Consumers of panel L



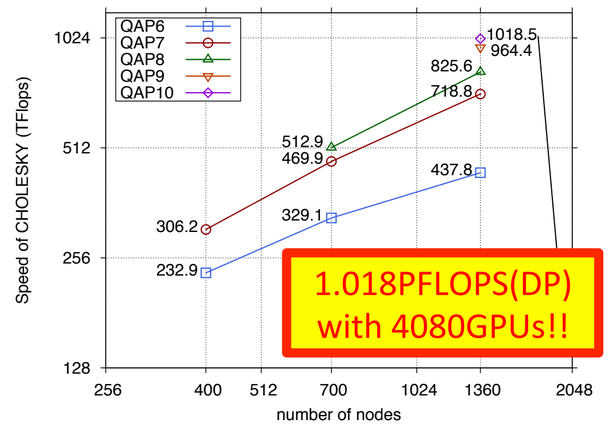
CPU/GPU Affinity

- 1 MPI process per GPU
  → 3 processes per node
- CPU/memory affinity is determined in a GPU centric way



Performance of CHOLESKY on TSUBAME2.0

Quadratic assignment problems (QAP)



SDPARA can solve the largest SDP problem
- DNN relaxation problem : QAP10 QAPLIB with 2.33 million constraints
- Using 1360 nodes, 2720 CPUs, 4080 M2050 GPUs
- 1.018PFLOPS in CHOLESKY (2.33m x 2.33m)
- The fastest and largest result as mathematical optimization problems!!