Implementation and Evaluation of 2.5D Matrix Multiplication on the K computer

Nov. 14, 2017
PRACE@SC17, Denver
RIKEN Advanced Institute for Computational Science
Daichi Mukunoki and Toshiyuki Imamura
2.5D matrix multiplication (2.5D-PDGEMM) [Solomonik & Demmel 2011]

- A communication-avoiding algorithm for parallel matrix-multiplication (PDGEMM) to improve the strong scaling performance
- On highly parallel systems, the performance of PDGEMM can become communication-bound when the problem size is not sufficiently large
- 2.5D-PDGEMM reduces the number of communications by utilizing data redundancy in a 2.5D distribution on a 3D process grid

1. Matrices with a 2D distribution are stacked vertically in a 3D process grid (2.5D distribution)
2. $1/c$ of 2D-PDGEMM algorithm is performed on each level; # of steps (and communications) is reduced
3. The final result is computed by vertically reducing the intermediate results on each level
Introduction (cont’d)

- **2D-compatible 2.5D-PDGEMM**
  - To compute matrices distributed in 2D on a 2D process grid, the matrices must be redistributed into 2.5D on a logical 3D process grid created on the 2D process grid.
  - 2D-compatible 2.5D-PDGEMM, which includes the matrix redistribution between 2D and 2.5D, is needed to be a substitute to the conventional PDGEMM.
  - Several studies have implemented and evaluated a 2.5D-PDGEMM, but such a 2D-compatible implementation has not been studied.

- **Contributions of this study**
  - Proposed a 2D-compatible 2.5D-PDGEMM implementation, that includes matrix redistribution between 2D and 2.5D.
  - Analyzed the performance using up to 16384 nodes on the K computer, by providing the performance breakdown + *performance estimation (new update)*
2D-compatible 2.5D-PDGEMM based on SUMMA

1. A logical 3D process grid is created on the initial 2D process grid, and vertical & horizontal sub-communicators are created using `MPI_Comm_split`
2. Matrices A & B are redistributed from 2D into 2.5D using `MPI_Allgather` on the vertical sub-communicators
3. Matrix multiplication is computed by 2.5D-SUMMA using `MPI_Bcast` on the horizontal sub-communicators
4. Matrix C is computed by reducing the temporal results on each level of the logical 3D process grid using `MPI_Allreduce` on the vertical sub-communicators
Execution time of our 2D-compatible 2.5D-PDGEMM

\[
T_{\text{Pdgemm25D}} = 2T_{\text{Allgather}}(c, en^2/p) + \sqrt{p/c^3}(2T_{\text{Bcast}}(\sqrt{p/c}, cen^2/p) + T_{\text{Dgemm}}(n/\sqrt{p/c})) + T_{\text{Allreduce}}(c, cen^2/p)
\]

Redistributions of mat.A&B
2.5D-SUMMA
Redistribution & Reduction of mat.C

On the K computer

\[
T_{\text{Bcast}}(p, m) = (p - 1)(l + s/b) + l + t_0 + (m/s) \max (l, s/b)
\]

\[
T_{\text{Allgather}}(p, m) = (p - 1)(l + gm/b) + l
\]

\[
T_{\text{Allreduce}} = T_{\text{Bcast}} + T_{\text{reduce}}
\]

\[
T_{\text{Reduce}}(p, m) = (p - 1)(l + s/b + s/b_{\text{comp}}) + l + (gm/s) \max (l + s/b_{\text{comp}}, s/b)
\]

\[
T_{\text{Dgemm}}(n) = 2n^3/s_{\text{Dgemm}}
\]

- \( p \): number of processes
- \( m \): message size
- \( l = 1.6 \, [\mu\text{sec}] \): MPI latency
- \( s = 16384 \, [\text{bytes}] \): segment size
- \( b = 3000 \, [\text{MB/sec}] \): bandwidth
- \( b_{\text{comp}} = 10000 \, [\text{MB/sec}] \): throughput of reduction
- \( t_0 = 8.37 \, [\text{sec}] \): overhead for selecting optimal algorithm (incl. one Allreduce)
- \( s_{\text{Dgemm}} = 115.2 \): Flops value of DGEMM
- \( g = 1.5\sqrt{c} \): effect of congestion
Performance Evaluation on the K computer

The K computer
- 10th fastest supercomputer in TOP500 as of Nov 2017
- 11.28 PFlops peak with 88128 nodes (this study used up to 16384 nodes of them)
- Each node has a SPARC64 VIIIfx (8-core, 128GFlops on double-precision)
- Torus fusion (Tofu) interconnect (6D mesh&torus, 5GB/s for each direction)

Evaluation
- We evaluated our implementation with different stack sizes: c=1, 4, and 16
  - When c=1, our code is equivalent to conventional 2D-SUMMA
- Even though the K computer has the 3D network topology, we run our programs as 2D process jobs in order to assume the 2D-compatible 2.5D-PDGEMM is used in a 2D application
Performance (strong scaling, n=32768)

Performance on the K computer (n=32768)

SUMMA(c=1) – measured
SUMMA(c=4) – measured
SUMMA(c=16) – measured
ScaLAPACK – measured
SUMMA(c=1) – estimated
SUMMA(c=4) – estimated
SUMMA(c=16) – estimated

Better

2.5D-PDGEMM improves the strong scaling performance even when the cost for matrix redistribution between 2D and 2.5D is included

† SUMMA(c=1) is equivalent to 2D-SUMMA
† MPI communicator setup cost is excluded
† ScaLAPACK was executed with nb=128
Performance Breakdown (strong scaling, n=32768)

Measured performance

- c=1
- c=4
- c=16

Estimated performance

- c=1
- c=4
- c=16

Horizontal communication (computation of SUMMA)
Vertical communication (redistribution & reduction)

†(L): latency factor (B): bandwidth factor
Summary

**Summary**
- “2D-compatible 2.5D-PDGEMM”, which is designed to perform computations of 2D distributed matrices on a 2D process grid
- Performance evaluation on the K computer, and performance analysis by providing the performance breakdown and performance estimation

**Conclusion**
- 2.5D-PDGEMM is effective in improving strong scaling performance even when the cost for matrix redistribution between 2D and 2.5D is included
- 2D-compatible implementation would be a good alternative to 2D-PDGEMM for highly parallel systems, in particular for the case when applications using the conventional PDGEMM are ported onto future systems that have greater parallelism
- The redistribution cost would be non-negligible when the stack size of the 2.5D algorithm increases; a tradeoff between the communication cost in the matrix multiplication and the redistribution & reduction costs

[Latest publication]