Assignment 2: SUMMA Matrix Multiplication Algorithm

Lars Karlsson (assistant)
Bo Kågström
Mikael Rännar

Due date: 2009-04-23 (17.00)

1 Introduction and Theory

Matrix multiplication is a common operation in dense linear algebra algorithms and it is important to optimize its performance. The aim of this assignment is to get practical experience with the 2D block cyclic data distribution and theoretical as well as experimental analysis of scalability. Your task is to implement a matrix multiplication algorithm known as SUMMA and evaluate the potential benefits of overlapping communication with computation.

The Basic Linear Algebra Subprograms (BLAS) is a standardized set of interfaces to common linear algebra operations. The double precision matrix multiply is a so called level-3 BLAS operation and the routine is called `DGEMM` (Double precision GEneral Matrix Multiply and add) in BLAS terminology and performs the set of operations described by

$$C \leftarrow \alpha \operatorname{op}(A) \operatorname{op}(B) + \beta C,$$

where $\operatorname{op}(X) = X$ or $\operatorname{op}(X) = X^T$ and α, β are double precision scalars. In other words, this means that A and/or B may be implicitly transposed.

In this assignment, we limit ourselves to

$$C \leftarrow AB + C.$$

The matrices A, B, and C have dimensions $m \times k$, $k \times n$, and $m \times n$, respectively. The dimension m is blocked with blocking factor m_b, n with n_b, and k with k_b. The matrices are distributed using a 2D block cyclic distribution and their first elements all map to process $(0, 0)$. Formally, the distribution onto a $P_r \times P_c$ process mesh is described by the following element-to-process mappings:

$$\begin{aligned}
A(i,j) &\mapsto (p,q), \quad p = \left\lfloor \frac{i}{m_b} \right\rfloor \mod P_r, \quad q = \left\lfloor \frac{j}{k_b} \right\rfloor \mod P_c, \\
B(i,j) &\mapsto (p,q), \quad p = \left\lfloor \frac{i}{k_b} \right\rfloor \mod P_r, \quad q = \left\lfloor \frac{j}{n_b} \right\rfloor \mod P_c, \\
C(i,j) &\mapsto (p,q), \quad p = \left\lfloor \frac{i}{m_b} \right\rfloor \mod P_r, \quad q = \left\lfloor \frac{j}{n_b} \right\rfloor \mod P_c.
\end{aligned}$$

See Figure 1 for an example (highlighted blocks are mapped to process $(1, 1)$).

2 Assignment

The SUMMA algorithm (without overlap) is as follows.

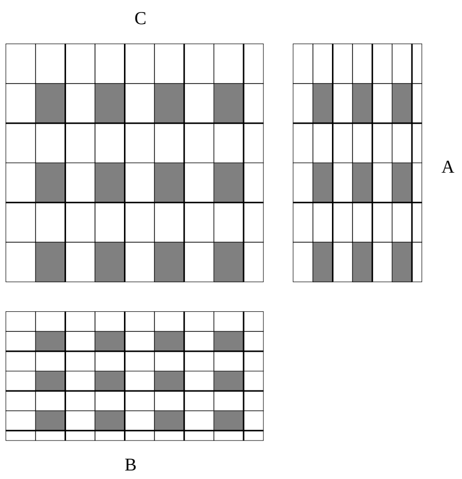

Figure 1: Distribution of the matrices on a 2-by-2 process mesh.

```
for j = 0 to k-1 step kb
   blksz = min(kb, k - j)
   Broadcast A(:, j:j+blksz-1) along mesh rows into the E buffer
   Broadcast B(j:j+blksz-1, :) along mesh columns into the S buffer
   Update C = C + E*S using local GEMM from BLAS library
end for
```

The algorithm steps through the block columns (rows) of A (B) and in each step a block outer product update on C is performed.

In Figure 2, we illustrate a snapshot of the algorithm as observed by the $(1, 1)$ process. The local part of C (light gray) is updated by using parts of the block column of A and block row of B (dark gray). The local view is for the $(1, 1)$ process only (dimensions of local matrices vary across the mesh). The relations between the dimensions of A and the dimensions of the S and E buffers are hopefully apparent from the figure.

Below is a list of some of the things we expect you to do. Remember that your task is to implement and evaluate the SUMMA algorithm using the concepts taught in the course as well as your own ideas.

- Implement the basic algorithm as well as a variant that overlaps communication with computation.
- Find T_p, S_p, E_p, and the asymptotic iso-efficiency function.
- Perform an *experimental* analysis of iso-efficiency.
- Is the algorithm scalable? Cost optimal? What is the memory constrained speedup of the algorithm?

3 Hints

The list below contains some hints to help you with the assignment.

1. On Akka we recommend these modules:

   ```
   module add mvapich/psc
   module add libgoto
   ```

   To link with GotoBLAS you need to add `-lgoto` to the linker command.

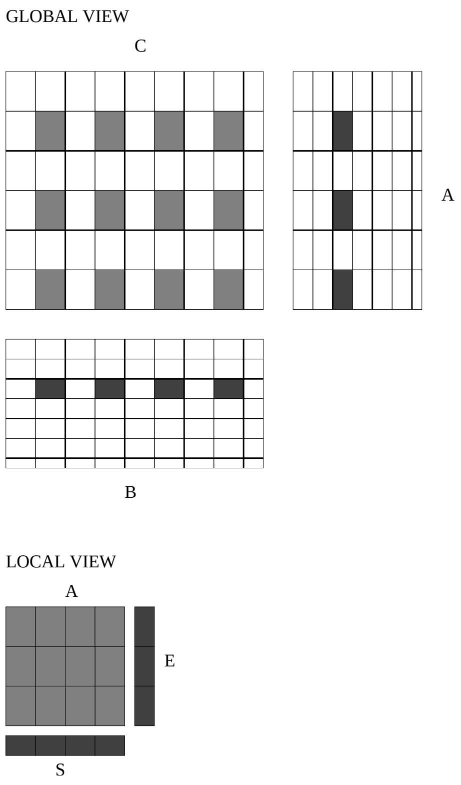

Figure 2: Snapshot of the algorithm at `j = 2*kb`.

2. Documentation for `DGEMM` is available at `http://www.netlib.org/blas/dgemm.f`.
3. If you call `DGEMM` from C you should be aware that interfacing Fortran from C is highly compiler specific. On Akka, this can be assumed:
   - the symbol for `DGEMM` is `dgemm_`,
   - all arguments must be pointers (`char*`, `double*`, `int*`) which in particular means that you can not pass the scalars α, β as literal constants.
4. The MPI v1.1 standard document in HTML format: `http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html`.
5. ScaLAPACK has several auxiliary routines that are useful when working with 2D block cyclic data layouts. Take a look at the `numroc` function and the `infog2l` subroutine at `http://www.netlib.org/scalapack/tools/`.