J Supercomput
DOI 10.1007/s11227-014-1133-x

Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms

Khalid Hasanov · Jean-Noël Quintin · Alexey Lastovetsky

Abstract Many state-of-the-art parallel algorithms, which are widely used in scientific applications executed on high-end computing systems, were designed in the twentieth century with relatively small-scale parallelism in mind. Indeed, while in 1990s a system with few hundred cores was considered a powerful supercomputer, modern top supercomputers have millions of cores. In this paper, we present a hierarchical approach to optimization of message-passing parallel algorithms for execution on large-scale distributed-memory systems. The idea is to reduce the communication cost by introducing hierarchy and hence more parallelism in the communication scheme. We apply this approach to SUMMA, the state-of-the-art parallel algorithm for matrix–matrix multiplication, and demonstrate both theoretically and experimentally that the modified Hierarchical SUMMA significantly improves the communication cost and the overall performance on large-scale platforms.

Keywords Matrix Multiplication · Parallel Computing · Exascale Computing · Communication Cost · Grid5000 · BlueGene · Hierarchy

1 Introduction

A significant proportion of scientific applications developed for execution on high-end computing systems are based on parallel algorithms proposed between the 1970s

K. Hasanov (✉) · A. Lastovetsky
University College Dublin, Belfield, Dublin 4, Ireland
e-mail: khalid.hasanov@ucdconnect.ie

A. Lastovetsky
e-mail: Alexey.Lastovetsky@ucd.ie

J.-N. Quintin
Extreme Computing R&D, Bull, France
e-mail: jean-noel.quintin@bull.net

Published online: 04 March 2014

and 1990s. These algorithms were designed with relatively small computing systems in mind and tested on such systems. Indeed, in 1995, the number of cores in the top 10 supercomputers ranged from 42 to 3,680 [1]. Nowadays, in June 2013, this number ranges from 147,456 to 3,120,000. Thus, over past two decades the number of processors in HPC systems has increased by three orders of magnitude. This drastic increase in scale significantly increases the cost of coordination and interaction of processes in traditional message-passing data-parallel applications. In other words, it increases their communication cost. In these applications, all processes are peers and the number of directly interacting processes grows quickly with the increase of their total number.

In this paper, we address the problem of reduction of the communication cost of such traditional message-passing data-parallel applications on large-scale distributed-memory computing systems. The approach we propose is a traditional methodology widely used for dealing with the complexity of coordination and management of a large number of actors, namely the hierarchical approach. According to this technique, thousands or millions of actors are structured, and instead of interacting with a large number of peers, they coordinate their activities with one superior and a small number of peers and inferiors. This way the overhead of interaction is significantly reduced. In the presented work, we demonstrate how this approach can be applied to optimization of the execution of parallel matrix–matrix multiplication on large-scale HPC platforms. We choose matrix multiplication for two reasons. First of all, it is important in its own right as a computational kernel of many scientific applications. Second, it is a popular representative for other scientific applications. It is widely accepted that if an optimization method works well for matrix multiplication, it will also work well for many other scientific applications.

The contributions of this paper are as follows:

- We introduce a new design to parallel matrix multiplication algorithm by introducing a two-level virtual hierarchy into the two-dimensional arrangement of processors. We apply our approach to the SUMMA algorithm [2], which is a state-of-the-art algorithm.
- We theoretically prove that hierarchical SUMMA (HSUMMA) reduces the communication cost of SUMMA and then provide experimental results on a cluster of Grid'5000 (a popular research infrastructure consisting of 20 clusters distributed over nine sites in France and one in Luxembourg) and BlueGene/P, which are reasonably representative and span a good spectrum of loosely and tightly coupled platforms. We compare HSUMMA only with SUMMA because it is the most general and scalable parallel matrix multiplication algorithm, which decreases its per-processor memory footprint with the increase of the number of processors for a given problem size, and is widely used in modern parallel numerical linear algebra packages such as ScaLAPACK. In addition, because of its practicality SUMMA is used as a starting point for implementation of parallel matrix multiplication on specific platforms. As a matter of fact, the most used parallel matrix multiplication algorithms for heterogeneous platforms [3,4] are based on SUMMA as well. Therefore, despite being introduced in the mid-1990s, SUMMA is still a state-of-the-art algorithm.

- Despite the study presented in this paper has been conducted in the context of parallel matrix multiplication, the proposed optimization technique is not application-bound and can be applied to other parallel applications intensively using broadcasts.

2 Previous work

In this section, we first outline existing optimizations of dense serial matrix multiplication algorithms and introduce parallel matrix multiplication algorithms on distributed-memory platforms. Then we detail the SUMMA algorithm, which is the algorithm of choice for our optimization. Finally, we briefly overview and discuss the existing broadcast algorithms, which can be used in parallel matrix multiplication algorithms to reduce their communication cost.

2.1 Serial matrix multiplication optimization

Matrix multiplication is a very important kernel in many numerical linear algebra algorithms and is one of the most studied problems in high-performance computing. Different approaches have been proposed to optimize matrix multiplication through the improvement of spatial and temporal locality. Blocking (or tiling) [5] is one such basic technique. Despite its generality, blocking is architecture-dependent. Cache-oblivious algorithms [6], on the other hand, offer an architecture-independent alternative to the blocked algorithms by using the divide-and-conquer paradigm. However, a recent study [7] shows that even highly optimized cache-oblivious programs perform considerably slower than their cache-conscious (i.e. based on blocking) counterparts. A related idea to the cache-oblivious methods is to use a recursive structure for the matrices [8]. However, traditional implementations of the Basic Linear Algebra Subroutines (BLAS) library [9] are mainly based on the blocking approach and thus need optimization on a specific hardware platform. Therefore, automatic optimization of matrix multiplication on a range of platforms has been an active research area. One such example is ATLAS [10] which provides C and Fortran77 interfaces to a portably efficient BLAS implementation and automatically generates optimized numerical software for a given processor architecture as a part of the software installation process. The GotoBLAS [11] library offers another high-performance implementation of matrix multiplication for a variety of architectures.

2.2 Parallel matrix multiplication optimization

Parallel matrix multiplication has also been thoroughly investigated over the past three decades. As a result, many parallel matrix multiplication algorithms have been developed for distributed memory, shared memory and hybrid platforms. In this section, we only outline the algorithms designed for distributed memory platforms.

Cannon's algorithm [12], which was introduced in 1967, was the first efficient algorithm for parallel matrix multiplication providing theoretically optimal communication cost. However, this algorithm requires a square number of processors which

makes it impossible to be used in a general purpose library. Fox's algorithm [13], which was introduced later, has the same restriction. PUMMA [14] and BiMMeR [15] extend Fox's algorithm for a general 2-D processor grid by using block-cyclic data distribution and torus-wrap data layout, respectively.

The 3D algorithm [16], which dates back to the 1990s, organizes the p processors as $p^{\frac{1}{3}} \times p^{\frac{1}{3}} \times p^{\frac{1}{3}}$ 3D mesh and achieves a factor of $p^{\frac{1}{6}}$ less communication cost than 2D parallel matrix multiplication algorithms. However, to get this improvement the 3D algorithm requires $p^{\frac{1}{3}}$ extra copies of the matrices. That means that on one million cores the $3D$ algorithm will require 100 extra copies of the matrices which would be a significant problem on some platforms. Therefore, the 3D algorithm is only practical for relatively small matrices.

Another method to improve the performance of parallel matrix multiplication is overlapping communication and computation. That approach was introduced by Agarwal et al. [17] in 1994 and according to the authors this optimization hides almost all of the communication cost with the computation for larger matrices.

In the mid-1990s, SUMMA [2] was introduced for a general $P \times Q$ processor grid. Like PUMMA and BiMMeR, SUMMA also solves the difficulty of Cannon's and Fox's algorithms and perfectly balances the computation load. However, SUMMA is simpler, more general and more efficient than the previous algorithms. For these reasons, it is used in ScaLAPACK [18], the most popular parallel numerical linear algebra package. The implementation of SUMMA in ScaLAPACK uses block-cyclic distribution and a modified communication scheme to overlap the communication and computation. The version of SUMMA modified this way was introduced as DIMMA [19]. Depending on the shape of the processor grid and matrix size, the performance of DIMMA can be better or worse than that of SUMMA. In its best case, the performance improvement of DIMMA over SUMMA was 10 % on Intel Paragon [19].

A more recent algorithm, SRUMMA [20], was proposed in 2004 and has algorithmic efficiency equivalent to that of Cannon's algorithm on clusters and shared memory systems. This algorithm uses block-checkerboard distribution of the matrices and overlaps communication with computations by using remote memory access (RMA) communication rather than message passing.

A recently introduced 2.5D algorithm [21] generalizes the 3D algorithm by parameterizing the extent of the third dimension of the processor arrangement: $\frac{p}{c}^{\frac{1}{2}} \times \frac{p}{c}^{\frac{1}{2}} \times c$, $c \in [1, p^{\frac{1}{3}}]$. While reducing the memory footprint compared with the 3D algorithm, it will still be efficient only if there is free amount of extra memory to store c copies of the matrices. At the same time, it is expected that exascale systems will have a dramatically shrinking memory space per core [22]. Therefore, the 2.5D algorithm cannot be scalable on future exascale systems.

2.3 SUMMA algorithm

SUMMA [2] implements the matrix multiplication $C = A \times B$ over a two-dimensional $p = s \times t$ processor grid. For simplicity, we assume that the matrices are square $n \times n$

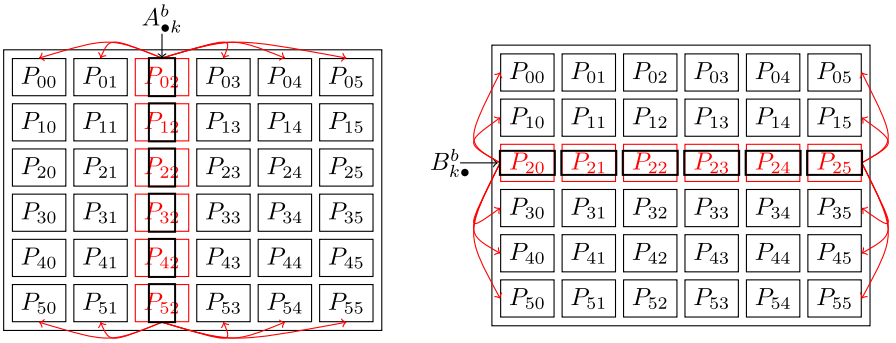

Fig. 1 Horizontal communications of matrix A and vertical communications of matrix B in SUMMA. The pivot column $A^b_{\bullet k}$ of $\frac{n}{\sqrt{P}} \times b$ blocks of matrix A is broadcast horizontally. The pivot row $B^b_{k\bullet}$ of $b \times \frac{n}{\sqrt{P}}$ blocks of matrix B is broadcast vertically

matrices. These matrices are distributed over the processor grid by block-checkerboard distribution.

We can see the size of the matrices as $\frac{n}{b} \times \frac{n}{b}$ by introducing a block of size b. Then each element in A, B and C is a square $b \times b$ block, and the algorithm operates on blocks rather than on scalar elements. For simplicity, we assume that n is a multiple of b. SUMMA can be formulated as follows: The algorithm consists of $\frac{n}{b}$ steps. At each step,

- each processor holding part of the pivot column of the matrix A horizontally broadcasts its part of the pivot column along the processor row.
- Each processor holding part of the pivot row of the matrix B vertically broadcasts its part of the pivot row along the processor column.
- Each processor updates each block in its C rectangle with one block from the pivot column and one block from the pivot row, so that each block c_{ij}, $(i, j) \in \left(1, \ldots, \frac{n}{b}\right)$ of matrix C will be updated as $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$.
- After $\frac{n}{b}$ steps of the algorithm, each block c_{ij} of matrix C will be equal to $\sum_{k=1}^{\frac{n}{b}} a_{ik} \times b_{kj}$.

Figure 1 shows the communication patterns in SUMMA on 6×6 processors grid.

2.4 MPI broadcast algorithms

Collective communications are key operations in parallel matrix multiplication algorithms. We have already seen that the communication pattern of SUMMA is based on broadcast and any improvement in the broadcast algorithm will improve the communication cost of SUMMA as well. Therefore, in this subsection we briefly outline existing broadcast algorithms.

A significant amount of research has been done in MPI [23] collective communications, and especially in MPI broadcast algorithms [24]. Early implementations of broadcast algorithms assumed homogeneous and fully connected networks. These implementations were based on simple binary or binomial trees and a couple of algo-

rithms have been introduced to be more effective for large message sizes and use the benefits of hierarchical networks by using pipelined trees [25] or recursive halving algorithms [26]. Some MPI broadcast algorithms are designed for specific topologies, such as mesh or hypercube [27], or for hierarchical communication networks [28]. The hierarchical implementation [28] splits an MPI broadcast over multiple hierarchies and uses different broadcast algorithms for different hirerarchies. Most of the recent MPI broadcast algorithms are developed for specific architectures, such as Blue Gene [29,30] and Infiniband [31,32].

The optimization technique developed in our study is not architecture or topology specific. It is a result of the holistic analysis of the communication cost of a parallel matrix multiplication algorithm. During its execution, the matrix multiplication algorithm performs a large number of broadcast operations, some of which are executed in parallel. In our design, we aimed to minimize the total communication cost of this application rather than the cost of the individual broadcasts. However, it has become clear that despite being developed in the context of a particular application, the resulting technique is not application-bound. Eventually, it improves the performance of the application by transforming the broadcast algorithms used in the application into a two-level hierarchy. As such, this optimization can be applied to any standalone broadcast algorithm or to any application using broadcast operations. It is worth noting that many existing broadcast algorithms are tree-based and hence hierarchical themselves. However, their communication trees have a uniform regular structure, for example, binomial or binary and transformation of these algorithms into a two-level hierarchy using our technique will break the regular structure of the resulting communication trees.

3 Hierarchical SUMMA

Let $p = s \times t$ processors be distributed over the same two-dimensional virtual processor grid as in SUMMA, the matrices be square $n \times n$ matrices, b be the block size. Let the distribution of the matrices be the same as in SUMMA. HSUMMA partitions the virtual $s \times t$ processor grid into a higher level $I \times J$ arrangement of rectangular groups of processors, so that inside each group there will be a two-dimensional $\frac{s}{I} \times \frac{t}{J}$ grid of processors. Figure 2 compares the arrangement of processors in SUMMA with HSUMMA. In this example a 6×6 grid of processors is arranged into two-level 3×3 grids of groups and 2×2 grid of processors inside a group. Let $P_{(x,y)(i,j)}$ denote the processor (i, j) inside the group (x, y). HSUMMA splits the communication phase of the SUMMA algorithm into two phases and consists of $\frac{n}{b}$ steps. The pseudocode for HSUMMA is Algorithm 1 and it can be summarized as follows:

- Horizontal broadcast of the pivot column of the matrix A is performed as follows:
  1. First, each processor $P_{(k,y)(i,j)}, k \in (1, \ldots, I)$ holding part of the pivot column of the matrix A horizontally broadcasts its part of the pivot column to the processors $P_{(k,z)(i,j)}, z \neq y, z \in (1, \ldots, I)$ in the other groups. (Line 6–9)
  2. Now, inside each group (x, y) processor $P_{(x,y)(i,j)}$ has the required part of the pivot column of the matrix A and it further horizontally broadcasts it to the processors $P_{(x,y)(i,c)}, c \neq j, c \in (1, \ldots, \frac{s}{I})$ inside the group. (Line 15–17)

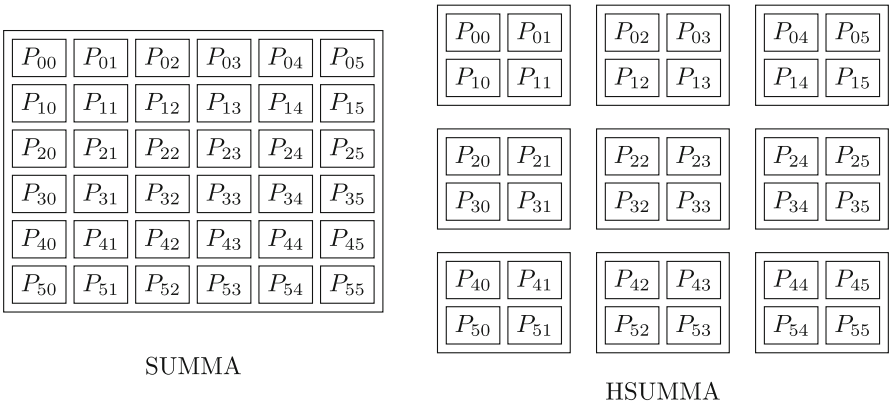

Fig. 2 SUMMA and HSUMMA. HSUMMA groups 6×6 processors into 3×3 groups, 2×2 processors per group

- Vertical broadcast of the pivot row of the matrix B is performed as follows:
  1. First, each processor $P_{(x,k)(i,j)}, k \in (1, \ldots, I)$ holding part of the pivot row of the matrix B vertically broadcasts its part of the pivot row to the processors $P_{(z,k)(i,j)}, z \neq k, z \in (1, \ldots, I)$ in the other groups. (Line 10–13)
  2. Now, inside each group (x, y) processor $P_{(x,y)(i,j)}$ has the required part of the pivot row of the matrix B and it further vertically broadcast it to the processors $P_{(x,y)(r,j)}, r \neq j, r \in (1, \ldots, \frac{t}{j})$ inside the group. (Line 18–20)
- Each processor inside a group updates each block in its C rectangle with one block from the pivot column and one block from the pivot row, so that each block $c_{ij}, (i, j) \in (1, \ldots, \frac{n}{b})$ of matrix C will be updated as $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$. (Line 21)
- After $\frac{n}{b}$ steps (Line 21) of the algorithm, each block c_{ij} of matrix C will be equal to $\sum_{k=1}^{\frac{n}{b}} a_{ik} \times b_{kj}$.

It is assumed that only one broadcast algorithm is used in all the steps of the algorithm and there is no barrier between the communications at the hierarchies. The communication phases described above are illustrated in Figs. 3 and 4. In general the block size between groups, M, and the block size inside a group, b, are different. In this case the size of sent data between the groups is at least the same as the size of data sent inside a group. Apparently, $b \leq M$. Then, the number of steps at the higher level will be equal to the number of blocks between groups: $\frac{n}{M}$. In each iteration between the groups, the number of steps inside a group will be $\frac{M}{b}$, so the total number of steps of HSUMMA, $\frac{n}{M} \times \frac{M}{b}$, will be the same as the number of steps of SUMMA. The amount of data sent will be also the same as in SUMMA.

In addition, SUMMA is a special case of HSUMMA. Indeed, when the number of groups, G, is equal to one or to the total number of processors, p, HSUMMA and SUMMA become equivalent. This means that even if there appears a highly efficient broadcast algorithm, the use of which makes SUMMA outperform HSUMMA for any $G \in (1, p)$, we should just use HSUMMA with $G = 1$.

Algorithm 1: Hierarchical SUMMA algorithm.

```
/*The A,B,C matrices are distributed on a virtual 2-D grid of p = s×t
processors.
Here are the instructions executed by the processor P(x,y)(i,j) (this is the
processor (i,j) inside the group (x,y)).*/
Data: NB_Block_Group: Number of steps in the higher level
Data: NB_Block_Inside: Number of steps in the lower level
Data: (M, L, N): Matrix dimensions
Data: A, B: two input sub-matrices of size (M/s × L/t, L/s × N/t)
Result: C: result sub-matrix of size M/s × N/t
begin
    /* Broadcast A and B                                                  */
1   MPI_Comm group_col_comm  /* communicator between P(*,y)(i,j) processors */
2   MPI_Comm group_row_comm  /* communicator between P(x,*)(i,j) processors */
3   MPI_Comm col_comm        /* communicator between P(x,y)(*,j) processors */
4   MPI_Comm row_comm        /* communicator between P(x,y)(i,*) processors */

5   for iter_group = 0; iter_group < NB_Block_Group; iter_group++ do

6       if i == Pivot_inside_group_col(iter_group) then
7           if x == Pivot_group_col(iter_group) then
                /* Get direct access to the iter^th_group group block of A  */
8               Copy_Block_group( Block_group_A, A, iter_group )
9           MPI_Bcast(Block_group_A, Type_Block_group_A, Pivot_group_col(iter_group),
            group_row_comm)

10      if j == Pivot_inside_group_row(iter_group) then
11          if y == Pivot_group_row(iter_group) then
                /* Get direct access to the iter^th_group group block of B  */
12              Copy_Block_group( Block_group_B, B, iter_group )
13          MPI_Bcast(Block_group_B, Type_Block_group_B, Pivot_group_row(iter_group),
            group_col_comm)

14      for iter = 0; iter < NB_Block_Inside; iter++ do
15          if i == Pivot_inside_group_col(iter) then
                /* Get access to the iter^th block of Block_group_A on this
                processor                                                  */
16              Copy_Block_A( Block_A, Block_group_A, iter)
17          MPI_Bcast(Block_A, Type_Block_A, Pivot_col(iter), row_comm)

18          if j == Pivot_inside_group_row(iter) then
                /* Get access to the iter^th block of Block_group_B on this
                processor                                                  */
19              Copy_Block_B( Block_B, Block_group_B, iter)
20          MPI_Bcast(Block_B, Type_Block_B, Pivot_row(iter), col_comm)

21          DGemm(Block_A, Block_B, C)
```

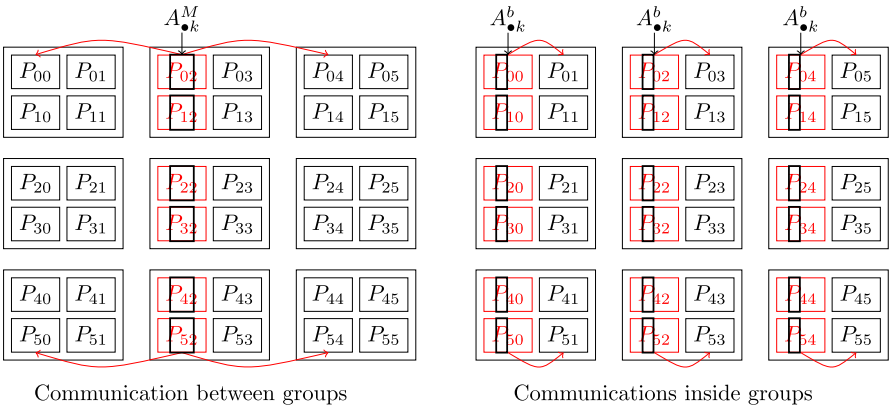

Fig. 3 Horizontal communications of matrix A in HSUMMA. The pivot column $A^M_{\bullet k}$ of $\frac{n}{\sqrt{P}} \times M$ blocks of matrix A is broadcast horizontally between groups. Upon receipt of the pivot column data from the other groups, the local pivot column $A^b_{\bullet k}$, $(b \leq M)$ of $\frac{n}{\sqrt{P}} \times b$ blocks of matrix A is broadcast horizontally inside each group

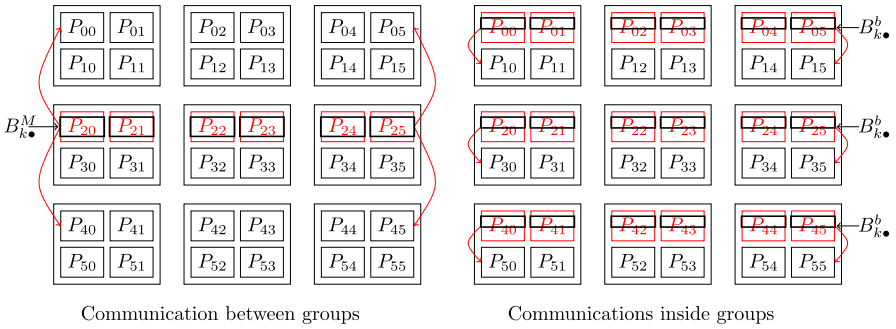

Fig. 4 Vertical communications of matrix B in HSUMMA. The pivot row $B^M_{k\bullet}$ of $M \times \frac{n}{\sqrt{P}}$ blocks of matrix B is broadcast vertically between groups. Upon receipt of the pivot row data from the other groups, the local pivot row $B^b_{k\bullet}$ of $b \times \frac{n}{\sqrt{P}}$, $(b \leq M)$ blocks of matrix B is broadcast vertically inside each group

4 Theoretical analysis

In this section SUMMA and HSUMMA are theoretically analyzed and compared. First of all, for simplicity we assume that the matrices are $n \times n$ square matrices. Let b be block size inside one group and M be block size between groups. As a communication model we use Hockney's model [33] which represents the time of sending of a message of size m between two processors as $\alpha + m\beta$. Here, α is the latency, and β is the reciprocal of the network bandwidth. In addition, let us assume that a combined floating point computation (for one addition and multiplication) time is γ. First, we analyze the cost of SUMMA and HSUMMA using two popular broadcast algorithms which are implemented in Open MPI and MPICH: the pipelined linear tree algorithm and the scatter-allgather algorithm. It is known that the costs of broadcasting a message of size m to p processors using these algorithm are as follows:

- Pipelined linear tree [25]: $(X + p - 1) \times \left(\alpha + \frac{m}{X} \times \beta\right)$
  Here, the height of the broadcast tree is equal to the number of processors, and the maximum nodal degree of the tree is one. According to the algorithm, a broadcast message of size m is split into X segments of size $\frac{m}{X}$, and X pipelined broadcasts of these segments will implement the operation.
- Scatter-allgather algorithm [24] : $(\log_2(p) + p - 1)\alpha + 2\frac{p-1}{p}m\beta$
  Here, scatter uses a binomial tree and allgather uses a ring algorithm in which the data from each processor are sent around a virtual ring of processors in $p - 1$ steps. MPICH broadcast implementation uses this algorithm for large messages and the number of processors greater than seven [24].

In addition, we use a generic broadcast model for general performance analysis of HSUMMA independent of the particular broadcast algorithm.

4.1 Analysis of SUMMA

Let the $n \times n$ matrices be distributed over a two-dimensional $\sqrt{p} \times \sqrt{p}$ grid of processors and let the block size be b. After distributing the matrices over the processors grid each processor will have a $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ part of the matrices. This algorithm has $\frac{n}{b}$ steps. In each step, the processors broadcast a pivot row of matrix B and a pivot column of matrix A. In our analysis, we assume that these two communication steps are serialized. The computation cost of one step is $O(2 \times \frac{n^2}{p} \times b)$. Hence, the overall computation cost will be $O(\frac{2n^3}{p})$.

For this analysis the network congestion is neglected. Broadcasting a pivot row (column) is broken down into a set of parallel broadcasts along the processor columns (rows). The size of data transferred by each such individual broadcast is $\frac{n}{\sqrt{p}} \times b$. The total communication cost of SUMMA can be computed by multiplying the communication cost of each step by the number of steps depending on the broadcast algorithm.

- The communication cost of broadcasting a pivot row or a pivot column with the pipelined linear tree broadcast in one step will be as follows:
  $\left(X + \sqrt{p} - 1\right) \times \left(\alpha + \beta \times \frac{n}{\sqrt{p}X} \times b\right)$
- The communication cost of broadcasting a pivot row or a pivot column with the scatter-algather broadcast in one step will be as follows:
  $\left(\log_2\left(\sqrt{p}\right) + \sqrt{p} - 1\right) \times \alpha + 2\left(1 - \frac{1}{\sqrt{p}}\right)\beta \times \frac{n}{\sqrt{p}} \times b$

If we sum the costs of the vertical and horizontal communications, and take into account that there are $\frac{n}{b}$ steps in total, then the overall communication costs will be as follows:

- Communication cost of SUMMA with the pipelined linear tree broadcast:

$$2\left(X + \sqrt{p} - 1\right) \times \left(\alpha \times \frac{n}{b} + \beta \times \frac{n^2}{\sqrt{p}X}\right)$$

- Communication cost of SUMMA with the scatter-allgather broadcast:

$$\left(\log_2(p) + 2(\sqrt{p} - 1)\right)\alpha \times \frac{n}{b} + 4\left(1 - \frac{1}{\sqrt{p}}\right)\beta \times \frac{n^2}{\sqrt{p}}.$$

4.2 Analysis of HSUMMA

To simplify the analysis, let us assume that there are G groups arranged as a $\sqrt{G} \times \sqrt{G}$ grid of processors groups. Let M denote the block size between groups (we also call such a block an outer block), b be the block size inside a group, and $n \times n$ be the size of the matrices.

HSUMMA has two communication phases: communication between groups (i.e. outer communication) and inside groups (i.e. inner communication). The outer communication phase has $\frac{n}{M}$ steps which are called outer steps. Each outer block belongs to $\sqrt{p}$ processors. Thus, in one outer step each processor, which owns a part of the pivot column, horizontally broadcasts this part (of size $\frac{n \times M}{\sqrt{p}}$) to $\sqrt{G}$ processors. Similarly, each processor, owning a part of the pivot row, will vertically broadcast its part (of size $\frac{n \times M}{\sqrt{p}}$) to $\sqrt{G}$ processors.

Inside one group, processors are arranged in a grid of size $\frac{\sqrt{p}}{\sqrt{G}} \times \frac{\sqrt{p}}{\sqrt{G}}$. Upon the receipt of the outer block, in the same way horizontal and vertical broadcasts are performed inside each group. The communications inside different groups happen in parallel as they are completely independent of each other. Inside a group there will be $\frac{M}{b}$ steps which we call inner steps. In each inner step, a data block of matrix A of size $\frac{n \times b}{\sqrt{p}}$ is broadcast horizontally to $\frac{\sqrt{p}}{\sqrt{G}}$ processors, and a data block of matrix B of size $\frac{n \times b}{\sqrt{p}}$ is broadcast vertically to $\frac{\sqrt{p}}{\sqrt{G}}$ processors. Upon the receipt of the required data, each processor updates its result by using a dgemm routine.

The total number of steps is $\frac{n}{b}$, and the overall computation cost again will be $O(\frac{2n^3}{p})$ as the computation cost in one inner step is $O(2 \times \frac{n^2}{p} \times b)$.

The overall communication cost inside a group will be the sum of the horizontal and vertical communication costs inside the group, multiplied by the number of inner steps. In the same way, the overall communication cost between the groups will be equal to the sum of the horizontal and vertical communication costs between the groups, multiplied by the number of outer steps. The total communication cost of HSUMMA will be the sum of the overall inner and outer communication costs. If we put the corresponding amount of communicated data and the number of communicating processors in the formulas for the costs of the pipelined linear tree algorithm and the scatter-allgather algorithm, the resulting communication costs will be as follows:

- Inner communication cost (inside groups):
  - Pipelined linear tree:

$$2\left(X + \sqrt{\frac{p}{G}} - 1\right) \times \left(\alpha \times \frac{n}{b} + \beta \times \frac{n^2}{\sqrt{p}X}\right)$$

- Scatter-allgather broadcast:

$$\left(\log_2\left(\frac{p}{G}\right)+2\left(\frac{\sqrt{p}}{\sqrt{G}}-1\right)\right)\times\alpha\times\frac{n}{b}+4\left(1-\frac{\sqrt{G}}{\sqrt{p}}\right)\times\frac{n^2}{\sqrt{p}}\beta$$

- Outer communication cost (between groups):
  - Pipelined linear tree:

$$2\left(X+\sqrt{G}-1\right)\times\left(\alpha\times\frac{n}{M}+\beta\times\frac{n^2}{\sqrt{p}X}\right)$$

  - Scatter-allgather broadcast:

$$\left(\log_2(G)+2\left(\sqrt{G}-1\right)\right)\times\alpha\times\frac{n}{M}+4\left(1-\frac{1}{\sqrt{G}}\right)\times\frac{n^2}{\sqrt{p}}\beta.$$

4.3 Theoretical prediction

One of the goals of this section is to demonstrate that independent of the broadcast algorithm employed by SUMMA, HSUMMA will either outperform SUMMA, or be at least equally fast. This section introduces a general model for broadcast algorithms and theoretically predicts SUMMA and HSUMMA. In the model we assume no contention and assume all the links are homogeneous. We show that even this simple model can predict the extremums of the communication cost function.

Again, we assume that the time taken to send a message of size m between any two processors is modeled as $T(m)=\alpha+m\times\beta$, where α is the latency and β is the reciprocal bandwidth.

We model a broadcast time for a message of size m among p processors by formula (1). This model generalizes all homogeneous broadcast algorithms, such as flat, binary, binomial, linear and scatter/allgather broadcast algorithms [25,34], which are used in state of the art broadcast implementations like MPICH [35] and Open MPI [36].

$$T_{bcast}(m,\,p)=L(p)\times\alpha+m\times W(p)\times\beta \tag{1}$$

In (1) we assume that $L(1)=0$ and $W(1)=0$. It is also assumed that $L(p)$ and $W(p)$ are monotonic and differentiable functions in the interval $(1,\,p)$ and their first derivatives are constants or monotonic in the interval $(1,\,p)$.

By using this general broadcast model the communication cost of HSUMMA can be expressed as a sum of the latency cost and the bandwidth cost:

$$T_{HS}(n,\,p,\,G)=T_{HS_l}(n,\,p,\,G)+T_{HS_b}(n,\,p,\,G) \tag{2}$$

Here $G \in [1, p]$ and $b \leq M$. The latency cost $T_{HS_l}(n, p, G)$ and the bandwidth cost $T_{HS_b}(n, p, G)$ will be given by the following formulas:

$$T_{HS_l}(n, p, G) = 2n\left(\frac{1}{M} \times L(\sqrt{G}) + \frac{1}{b} \times L\left(\frac{\sqrt{p}}{\sqrt{G}}\right)\right)\alpha \tag{3}$$

$$T_{HS_b}(n, p, G) = 2\frac{n^2}{\sqrt{p}} \times \left(W(\sqrt{G}) + W\left(\frac{\sqrt{p}}{\sqrt{G}}\right)\right)\beta \tag{4}$$

If we take $b = M$ the latency cost $T_{HS_l}(n, p, G)$ changes and becomes as follows:

$$T_{HS_l}(n, p, G) = 2n\left(\frac{1}{M} \times L(\sqrt{G}) + \frac{1}{M} \times L\left(\frac{\sqrt{p}}{\sqrt{G}}\right)\right)\alpha \tag{5}$$

However, the bandwidth cost will not change as it does not depend on the block sizes.

The comparison of Formula 3 and Formula 5 suggests that with decrease of b the latency cost will increase. This means that $b = M$ will be the optimal value for b. We will validate this prediction in the experimental part. Therefore, in the following analysis we take $M = b$.

It is clear that $T_S(n, p)$ (i.e. SUMMA) is a special case of $T_{HS}(n, p, G)$ (i.e. HSUMMA) when $G = 1$ or $G = p$.

Let us investigate extremums of T_{HS} as a function of G for fixed p and n. Then, for $M = b$ we can get the following derivatives:

$$\frac{\partial T_{HS}}{\partial G} = \frac{n}{b} \times L_1(p, G)\alpha + \frac{n^2}{\sqrt{p}} \times W_1(p, G)\beta \tag{6}$$

Here, $L_1(p, G)$ and $W_1(p, G)$ are defined as follows:

$$L_1(p, G) = \left(\frac{\partial L(\sqrt{G})}{\partial \sqrt{G}} \times \frac{1}{\sqrt{G}} - \frac{\partial L\left(\frac{\sqrt{p}}{\sqrt{G}}\right)}{\partial \frac{\sqrt{p}}{\sqrt{G}}} \times \frac{\sqrt{p}}{G\sqrt{G}}\right) \tag{7}$$

$$W_1(p, G) = \left(\frac{\partial W(\sqrt{G})}{\partial \sqrt{G}} \times \frac{1}{\sqrt{G}} - \frac{\partial W\left(\frac{\sqrt{p}}{\sqrt{G}}\right)}{\partial \frac{\sqrt{p}}{\sqrt{G}}} \times \frac{\sqrt{p}}{G\sqrt{G}}\right) \tag{8}$$

It can be easily shown that, if $G = \sqrt{p}$ then $L_1(p, G) = 0$ and $W_1(p, G) = 0$, thus, $\frac{\partial T_{HS}}{\partial G} = 0$. In addition, $\frac{\partial T_{HS}}{\partial G}$ changes the sign in the interval $(1, p)$ depending on the value of G. That means that $T_{HS}(n, p, G)$ has extremum at $G = \sqrt{p}$ for fixed n and p. The expression of $\frac{\partial T_{HS}}{\partial G}$ shows that, depending on the ratio of α and β the extremum can be either minimum or maximum in the interval $(1, p)$. If $G = \sqrt{p}$ is the minimum point it means that with $G = \sqrt{p}$ HSUMMA will outperform SUMMA; otherwise, HSUMMA with $G = 1$ or $G = p$ will have the same performance as SUMMA.

Now let us apply this analysis to the HSUMMA communication cost function obtained for *scatter-allgather* broadcast algorithm (see Sect. 4.2) again assuming

$b = M$ for simplicity. We will have

$$\frac{\partial T_{HS}}{\partial G} = \frac{G - \sqrt{p}}{G\sqrt{G}} \times \left(\frac{n\alpha}{b} - 2\frac{n^2}{p} \times \beta \right) \tag{9}$$

It is clear that if $G = \sqrt{p}$ then $\frac{\partial T_{HS}}{\partial G} = 0$. Depending on the ratio of α and β, the communication cost as a function of G has either minimum or maximum in the interval $(1, p)$.

- If

$$\frac{\alpha}{\beta} > 2\frac{nb}{p} \tag{10}$$

  then $\frac{\partial T_{HS}}{\partial G} < 0$ in the interval $(1, \sqrt{p})$ and $\frac{\partial T_{HS}}{\partial G} > 0$ in $(\sqrt{p}, p)$. Thus T_{HS} has the minimum in the interval $(1, p)$ and the minimum point is $G = \sqrt{p}$.
- If

$$\frac{\alpha}{\beta} < 2\frac{nb}{p}, \tag{11}$$

  then T_{HS} has the maximum in the interval $(1, p)$ and the maximum point is $G = \sqrt{p}$. The function gets its minimum at either $G = 1$ or $G = p$.

If we take $G = \sqrt{p}$ in the HSUMMA communication cost function (see Sect. 4.2) and assume the above conditions, the optimal communication cost function will be as follows:

$$\left(\log_2(p) + 4\left(\sqrt[4]{p} - 1\right)\right) \times \frac{n}{b} \times \alpha + 8\left(1 - \frac{1}{\sqrt[4]{p}}\right) \times \frac{n^2}{\sqrt{p}} \times \beta \tag{12}$$

We will use the scatter-allgather model to predict the performance on future exascale platforms.

Now, let us take the communication cost function of HSUMMA with the *pipelined-linear* tree broadcast(see Sect. 4.2) and find the extremum of the function in $(1, p)$.

$$\frac{\partial T_{HS}}{\partial G} = \frac{G - \sqrt{p}}{G\sqrt{G}} \times \left(\frac{n}{b} \times \alpha + \frac{n^2}{\sqrt{p}X} \times \beta \right) \tag{13}$$

In the same way it can be proved that with the pipelined linear tree broadcast, independent of α and β, $G = \sqrt{p}$ is the minimum point of the communication function in $(1, p)$. A theoretical analysis of HSUMMA with the binomial tree broadcast can be found in [37].

4.4 Prediction on Exascale

We use parameters obtained from a recent report on exascale architecture roadmap [38] to predict performance of HSUMMA on exascale platforms. Figure 5 shows that, theoretically, HSUMMA with any number of groups outperforms SUMMA. It is worth mentioning that if the number of groups is equal to 1 or p, then HSUMMA will be equivalent to SUMMA, as in that case there is no hierarchy. Thus, theoretically, the

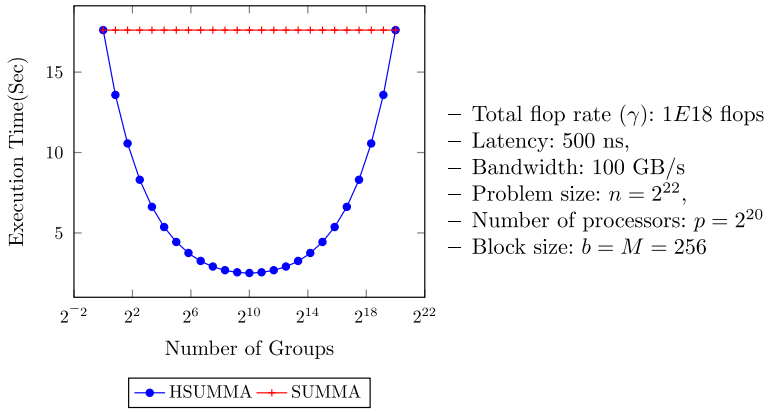

Fig. 5 Prediction of SUMMA and HSUMMA on Exascale. $p = 1{,}048{,}576$

communication cost function of HSUMMA has a parabola-like shape. In the following sections we will see that experimental results validate this theoretical prediction.

5 Experiments

Our experiments were carried out on a cluster of Grid'5000 [39] and a BlueGene/P (BG/P) platform which are fairly representative and span a good spectrum of loosely and tightly coupled platforms. The details of the platforms are given in the appropriate sections. The times in our experimental results are the mean time of 30 experiments.

5.1 Experiments on Grid'5000

Some of our experiments were carried out on the Grid'5000 infrastructure in France. The platform consists of 20 clusters distributed over 9 sites in France and one in Luxembourg. All sites are interconnected by 10 Gb/s high-speed network, except Reims, which is connected through a 1-Gb/s link. Each site consists of different technologies and clusters. Our experiments were performed on the Nancy site which is composed of three clusters: Graphene, Griffon and Graphite. We used the Graphene cluster for the experiments. The cluster is equipped with 144 nodes and each node has a disk of 320 GB storage, 16 GB of memory and 4-cores of CPU Intel Xeon X3440. The nodes in the Graphene cluster have one 20GB Infiniband and are interconnected via Gigabyte Ethernet. The Grid'5000 web site [39] provides more detailed information about the platform. We used multi-threaded dgemm from the GotoBlas2 library [11] for the sequential operations, MPICH 3.0.1 [35] and OpenMPI 1.4.5 [36] for MPI implementation and our implementations of the matrix multiplication algorithms. The size of the matrices in our experiments on Grid'5000 was $8,192 \times 8,192$. The experiments with OpenMPI have been done with both Ethernet and Infiniband networks.

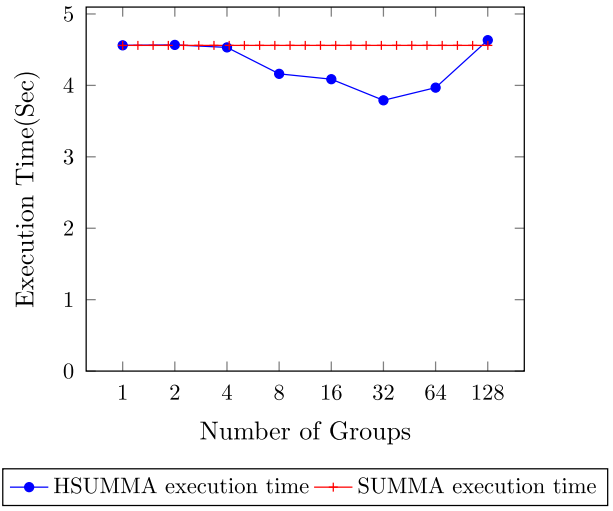

Fig. 6 Experiments with OpenMPI on G5000 with Ethernet. $b = M = 256$, $n = 8,192$ and $p = 128$

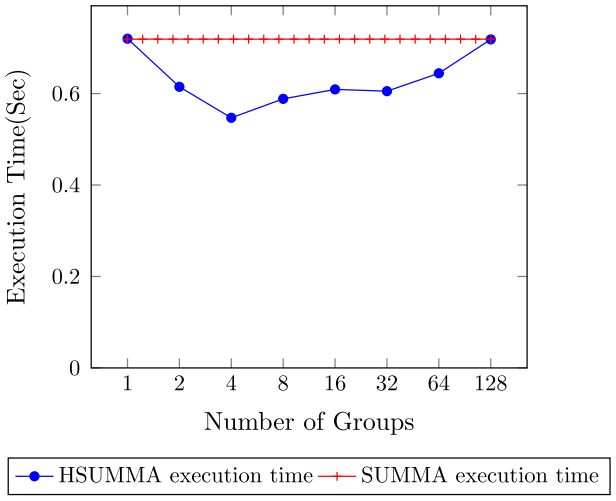

Fig. 7 Experiments with OpenMPI on G5000 with Infiniband. $b = M = 256$, $n = 8,192$ and $p = 128$

Here, we are not trying to compare different MPI implementations. Instead, we show that the benefit of HSUMMA over SUMMA does not depend on the MPI implementation.

Figure 6 shows that HSUMMA reduces the execution time of SUMMA by 16.8 percent on 128 nodes with an Ethernet network. The improvement with an Infiniband network is 24 percent (see Fig. 7). On the other hand, the improvement with MPICH is 7.75 times with block size 64 (see Fig. 8) and 2.96 times with block size 256 (see Fig. 9). This big difference comes from the MPI broadcast algorithm selection in MPICH depending on the message size and the number of processes. We did not fix the

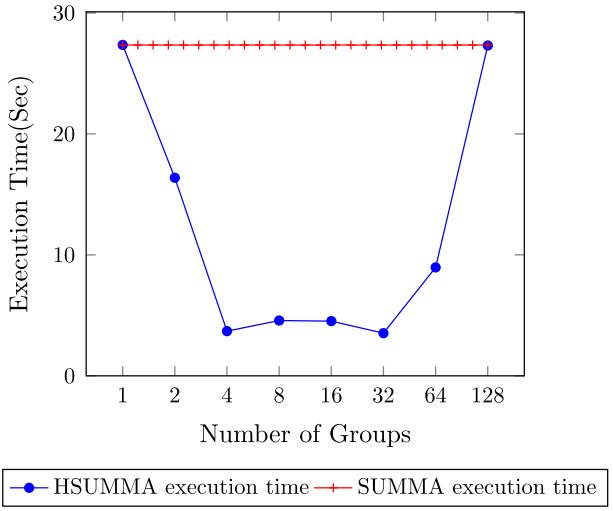

Fig. 8 Experiments with MPICH on G5000 with Ethernet. $b = M = 64$, $n = 8{,}192$ and $p = 128$

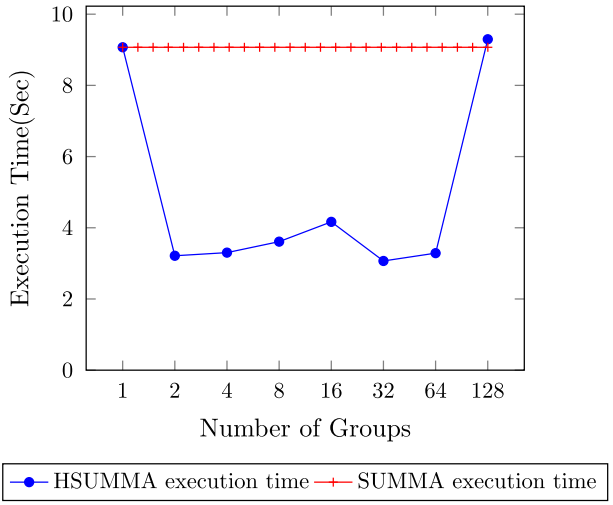

Fig. 9 Experiments with MPICH on G5000 with Ethernet. $b = M = 256$, $n = 8{,}192$ and $p = 128$

broadcast algorithm and allowed MPICH to decide which one to use. In these experiments, the default values of MPICH parameters (e.g. BCAST_SHORT_MSG_SIZE, BCAST_MIN_PROCS) [26] were used. We have also conducted experiments with a fixed broadcast algorithm (binomial tree, binary tree, flat tree, etc.). In all these experiments, we observed similar speedups.

We do not have the optimal number of groups exactly at $G = \sqrt{p}$. However, this does not downgrade our theoretical predictions as the shape of the cost function is similar to the theoretical shape.

5.2 Experiments on BlueGene/P

Some of our experiments were carried out on the Shaheen BlueGene/P at the Supercomputing Laboratory at King Abdullah University of Science&Technology (KAUST) in Thuwal, Saudi Arabia. Shaheen is a 16-rack BlueGene/P. Each node is equipped with four 32-bit, 850 Mhz PowerPC 450 cores and 4GB DDR memory. VN (Virtual Node) mode with torus connection was used for the experiments. The Blue Gene/P architecture provides a three-dimensional point-to-point Blue Gene/P torus network which interconnects all compute nodes and global networks for collective and interrupt operations. Use of this network is integrated into the BlueGene/P MPI implementation.

All the sequential computations in our experiments were performed by using the DGEMM routine from the IBM ESSL library. We have implemented SUMMA with block-checkerboard and block-cyclic distributions for comparison with HSUMMA. However, the data distribution in SUMMA does not change its performance on the BG/P. It may improve its performance if a modified communication pattern is used, as proposed in the DIMMA [19] algorithm. DIMMA was implemented in ScaLAPACK as a slight optimization of SUMMA; therefore, we also use ScaLAPACK (version 1.8.0) for the comparison with HSUMMA.

The benefit of HSUMMA comes from the optimal number of groups. Therefore, it is interesting to see how different numbers of groups affect the communication cost of HSUMMA on a large platform. Figure 10 shows HSUMMA on 16384 cores. In order to have a fair comparison again we use the same block size inside a group and between the groups. The figure shows that the execution time of SUMMA is 50.2 seconds. On the other hand, the minimum execution time of HSUMMA is 21.26 when $G = 512$. Thus, the execution time of HSUMMA is 2.36 times less than that of SUMMA on 16,384 cores. It is worth noting that different number of groups in HSUMMA does not affect the computation time, so all these reductions in the execution time come solely from the reduction of the communication time. In addition, according to our experiments, the improvement is 1.2 times on 2,048 cores and the performance of HSUMMA and SUMMA is almost the same on BlueGene/P cores smaller than 2,048.

The zigzags in Fig. 10 can be explained by the fact that mapping communication layouts to network hardware on BlueGene/P impacts the communication performance, and which was observed by Balaji et al. [40] as well. When we group processors we do not take into account the network topology. However, according to our preliminary observations these zigzags can be eliminated by taking the topology into account while grouping. Figure 11 represents scalability comparison of HSUMMA with SUMMA from communication point of view. Here, we use SUMMA both with block-checkerboard and block-cyclic distributions. It can be seen that HSUMMA is more scalable than both block-checkerboard and block-cyclic SUMMA, and this pattern suggests that the communication performance of HSUMMA rapidly improves compared to that of SUMMA as the number of cores increases.

According to the theoretical predictions, with some application/platform settings HSUMMA may not reduce the communication cost of SUMMA. We experimentally observed these phenomena on a smaller number of cores on the BG/P. Figure 12 illustrates one such experiment on 1,024 cores, where the best performance of HSUMMA was achieved with $G = 1$ and $G = p$. In this experiment, the interconnect type used

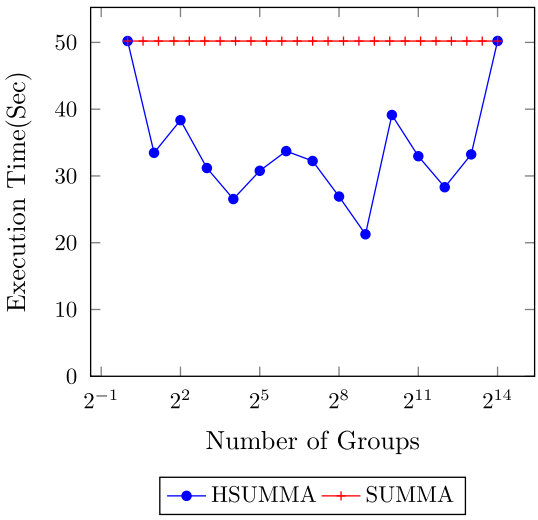

Fig. 10 Execution time of SUMMA and HSUMMA on 16,384 cores on BG/P. $b = M = 256$ and $n = 65{,}536$

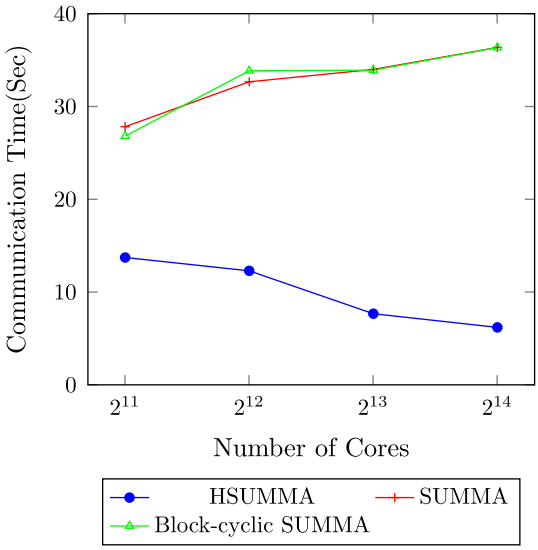

Fig. 11 Communication time of SUMMA, block-cyclic SUMMA and HSUMMA on BG/P. $b = M = 256$, $n = 65{,}536$

between base partitions of the BG/P was a mesh as the minimum number of cores to use a torus interconnect is 2,048.

5.3 Effect of different block sizes

In Sect. 4.3, it has been theoretically proven that the increase of the block size inside the groups will decrease the communication cost of HSUMMA. This section presents experimental results validating this prediction.

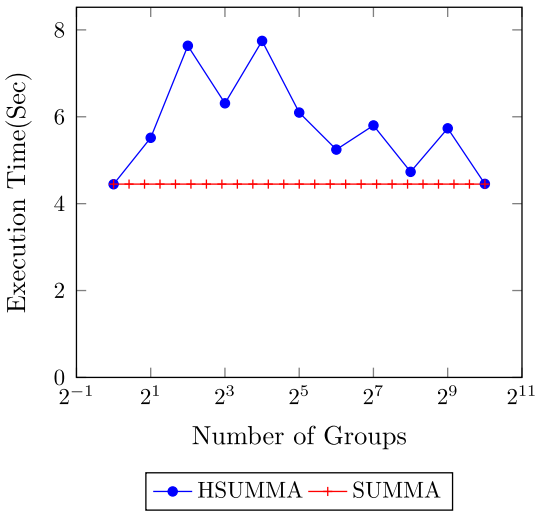

Fig. 12 Execution time of HSUMMA and SUMMA on 1,024 cores on BG/P. $b = M = 256$, $n = 16{,}384$

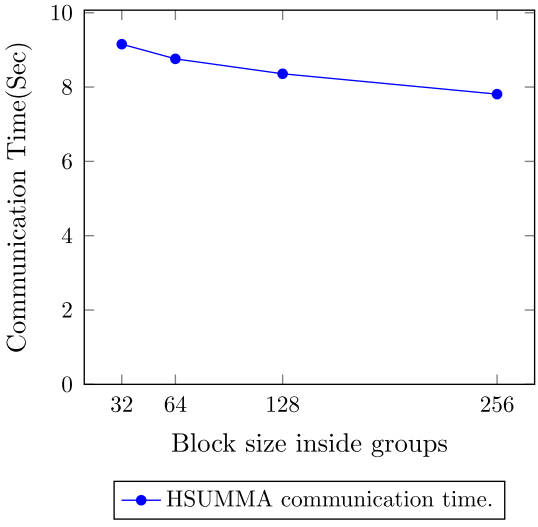

Fig. 13 HSUMMA on BG/P. $M = 256$, $G = 4$, $p = 16{,}384$ and $n = 65{,}536$

Figure 13 shows experimental results of HSUMMA with different block sizes inside the groups for the block size between the groups fixed to 256 and the number of groups fixed to 4. It can be seen that the communication time slightly decreases as the block size increases. Another interesting result is that, according to Fig. 14, the relative performance of HSUMMA for different numbers of groups does not depend on the block size inside the groups. In particular, this means that the optimal value of G does not depend on the block size inside the groups, and, therefore, any block size can be used in the procedure searching for the optimal value of G.

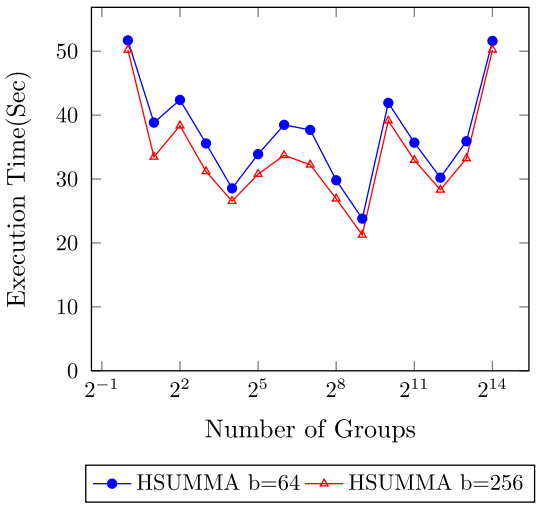

Fig. 14 HSUMMA on 16,384 cores on BG/P. $M = 256$ and $n = 65{,}536$

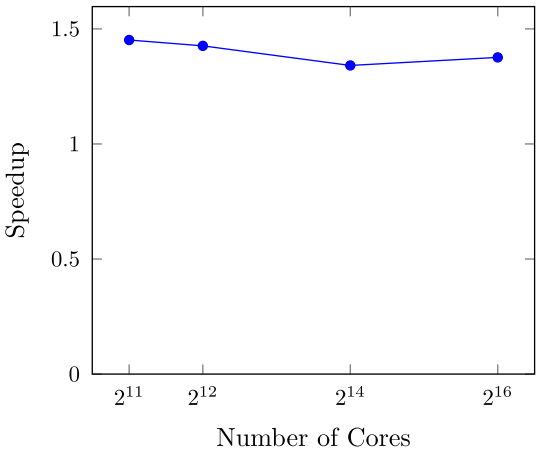

Fig. 15 Speedup of HSUMMA over ScaLAPACK on BG/P. $b = M = 256$ and $n = 65{,}536$

5.4 Comparison with ScaLAPACK

This section compares HSUMMA with the PDGEMM routine from the ScaLAPACK (ver. 1.8.0) library. The results of the corresponding experiments are shown in Fig. 15. Unfortunately, IBM PESSL is not available on the BG/P and, therefore, we cannot provide experimental results with PDGEMM from the PESSL library. However, it is known [41] that, unlike LU decomposition, PDGEMM from PESSL does not have any improvement over PDGEMM from ScaLAPACK. Moreover, the ScaLAPACK library on the BG/P uses a DGEMM from the IBM ESSL library which is optimized for Blue Gene.

6 Conclusions

We can conclude that our two-level hierarchical approach to parallel matrix multiplication significantly reduces the communication cost on large platforms such as BlueGene/P. The experiments show that HSUMMA achieves 2.08 times and 5.89 times less communication time than SUMMA on 2, 048 cores and on 16, 384 cores, respectively. Moreover, the overall execution time of HSUMMA is 1.2 times less than the overall execution time of SUMMA on 2, 048 cores and 2.36 times less on 16, 384 cores. This trend suggests that, while the number of processors increases, HSUMMA will be more scalable than SUMMA. In addition, our experiments on Grid'5000 show that HSUMMA can be effective on small platforms as well.

We select the optimal number of groups sampling over valid values. However, it can be easily automated and incorporated into the implementation by using a few iterations of HSUMMA.

HSUMMA is an extension of SUMMA, becoming SUMMA if $G = 1$ or $G = p$. Therefore, SUMMA cannot outperform HSUMMA. In the worst case scenario, they will have the same performance.

Despite our optimization has been developed in the context of parallel matrix multiplication, it can be applied to any standalone broadcast algorithm and any application intensively using broadcasts.

Acknowledgments The research in this paper was supported by IRCSET (Irish Research Council for Science, Engineering and Technology) and IBM, grant numbers EPSG/2011/188 and EPSPD/2011/207. Some of the experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr) Another part of the experiments was carried out using the resources of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) in Thuwal, Saudi Arabia. The authors would like to thank Ashley DeFlumere for her useful comments and corrections.

References

1. Top 500 supercomputer sites. http://www.top500.org/
2. van de Geijn RA, Jerrell W (1997) SUMMA: scalable universal matrix multiplication algorithm. Concurr Pract Exp 9(4):255–274
3. Beaumont O, Boudet V, Rastello F, Robert Y (2001) Matrix multiplication on heterogeneous platforms. IEEE Trans Parallel Distrib Syst 12(10):1033–1051
4. Lastovetsky A, Dongarra J (2009) High performance heterogeneous computing. Wiley, New York
5. Gustavson FG (2012) Cache blocking for linear algebra algorithms. Parallel processing and applied mathematics. In: Lecture Notes in Computer Science, vol 7203. Springer, Berlin, pp 122–132
6. Frigo M, Leiserson CE, Prokop H, Ramachandran S (1999) Cache-oblivious algorithms. In: Proceedings of the 40th annual symposium on foundations of computer science, FOCS '99. IEEE Computer Society, Washington, DC, USA, p 285
7. Yotov K, Roeder T, Pingali K, Gunnels J, Gustavson F (2007) An experimental comparison of cache-oblivious and cache-conscious programs. In: Proceedings of the nineteenth annual ACM symposium on parallel algorithms and srchitectures., SPAA '07ACM, New York, NY, USA, pp 93–104
8. Chatterjee S, Lebeck AR, Patnala PK, Mithuna T (2002) Recursive array layouts and fast matrix multiplication. IEEE Trans Parallel Distrib Syst 13(11):1105–1123
9. Basic Linear Algebra Routines (BLAS). http://www.netlib.org/blas/

10. Clint WR, Dongarra JJ (1998) Automatically tuned linear algebra software. Proceedings of the 1998 ACM/IEEE conference on supercomputing. Supercomputing '98IEEE Computer Society, Washington, DC, USA, pp 1–27
11. Goto K, van De Geijn RA (2008) Anatomy of high-performance matrix multiplication. ACM Trans Math Softw 34(3):1–25
12. Cannon LE (1969) A cellular computer to implement the Kalman filter algorithm. Ph.D. thesis, Bozeman, MT, USA
13. Fox GC, Otto SW, Hey AJG (1987) Matrix algorithms on a hypercube I: matrix multiplication. Parallel Comput 4(1):17–31
14. Jaeyoung C, Walker DW, Dongarra J (1994) PUMMA: parallel universal matrix multiplication algorithms on distributed memory concurrent computers. Concurr Pract Exp 6(7):543–570
15. Huss-Lederman S, Jacobson E, Tsao A, Zhang G (1994) Matrix multiplication on the Intel Touchstone Delta. Concurr Pract Exp 6(7):571–594
16. Agarwal RC, Balle SM, Gustavson FG, Joshi M, Palkar P (1995) A three-dimensional approach to parallel matrix multiplication. IBM J Res Dev 39(5):575–582
17. Agarwal RC, Gustavson FG, Zubair M (1994) A High-performance matrix-multiplication algorithm on a distributed-memory parallel computer, using overlapped communication. IBM J Res Dev 38(6):673–681
18. Blackford LS, Choi J, Cleary A, D'Azeuedo E, Demmel J, Dhillon I, Hammarling S, Henry G, Petitet A, Stanley K, Walker D, Whaley RC (1997) ScaLAPACK user's guide. Society for industrial and applied mathematics, Philadelphia
19. Jaeyoung C (1997) A new parallel matrix multiplication algorithm on distributed-memory concurrent computers. In: High Performance Computing on the Information Superhighway, 1997. HPC, Asia '97, pp 224–229
20. Krishnan M, Nieplocha J (2004) SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems. In: Proceedings of parallel and distributed processing symposium
21. Solomonik E, Demmel J (2011)Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In: Euro-Par (2), Lecture Notes in Computer Science, vol 6853. Springer, Berlin, pp 90–109
22. U.S.Department of Energy: Exascale Programming Challenges. ASCR Exascale Programming Challenges Workshop (2011)
23. Message passing interface forum. http://www.mpi-forum.org/
24. Barnett M, Gupta S, Payne DG, Shuler L, Robert A, van de Geijn, Watts J (1994) Interprocessor collective communication library (InterCom). In: Proceedings of the scalable high performance computing conference. IEEE Computer Society Press, New York, pp 357–364
25. Patarasuk P, Yuan X, Faraj A (2008) Techniques for pipelined broadcast on ethernet switched clusters. J Parallel Distrib Comput 68(6):809–824
26. Thakur R, Rabenseifner R, Gropp W (2005) Optimization of collective communication operations in MPICH. Int J High Perform Comput Appl 19(1):49–66
27. Scott DS (1991) Efficient all-to-all communication patterns in hypercube and mesh topologies. In: Proceedings of the sixth conference distributed memory computing, pp 398–403
28. Graham RL, Venkata MG, Ladd J, Shamis P, Rabinovitz I, Filipov V, Shainer G (2011) Cheetah: a framework for scalable hierarchical collective operations. CCGRID, pp 73–83
29. Almási G, Heidelberger P, Archer CJ, Martorell X, Erway CC, Moreira JE, Steinmacher-Burow B, Zheng Y (2005) Optimization of MPI collective communication on BlueGene/L systems. In: Proceedings of the 19th annual international conference on supercomputing., ICS '05ACM, New York, NY, USA, pp 253–262
30. Kumar S, Dozsa G, Almasi G, Heidelberger P, Chen D, Giampapa ME, Blocksome M, Faraj A, Parker J, Ratterman J, Smith B, Archer CJ (2008) The deep computing messaging framework: generalized scalable message passing on the Blue Gene/P supercomputer. In: Proceedings of the 22nd annual international conference on supercomputing., ICS '08ACM, New York, NY, USA, pp 94–103
31. Hoefler T, Siebert C, Rehm W (2007) A practically constant-time MPI broadcast algorithm for large-scale InfiniBand clusters with multicast. In: IPDPS, IEEE, New York, pp 1–8
32. Liu J, Wu J, Panda DK (2004) High performance RDMA-based MPI implementation over InfiniBand. Int J Parallel Progr 32(3):167–198
33. Hockney RW (1994) The communication challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Comput 20(3):389–398

34. Pješivac-Grbović J (2007) Towards Automatic and Adaptive Optimizations of MPI Collective Operations. Ph.D. thesis, University of Tennessee, Knoxville
35. MPICH-A Portable Implementation of MPI. http://www.mpich.org/
36. Gabriel E, Fagg G, Bosilca G, Angskun T, Dongarra J, Squyres J, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain R, Daniel D, Graham R, Woodall T (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, pp 97–104
37. Quintin J., Hasanov K, Lastovetsky A (2013) Hierarchical parallel matrix multiplication on large-scale distributed memory platforms. In: 42nd International conference on parallel processing (ICPP 2013). IEEE, New York, pp 754–762
38. Kondo M (2012) Report on Exascale Architecture. In: IESP Meeting, Japan
39. Grid5000. http://www.grid5000.fr
40. Balaji P, Gupta R, Vishnu A, Beckman P (2011) Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems. Comput Sci R D 26(3–4):247–256
41. Blackford LS, Whaley RC (1998) ScaLAPACK Evaluation and Performance at the DoD MSRCs. Tech. Rep. LAPACK Working Note No. 136, Technical Report UT CS-98-388, University of Tennessee, Knoxville, TN (1998)