Matrix multiplication on multidimensional torus networks

Edgar Solomonik
James Demmel

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-28
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-28.html

February 22, 2012

Acknowledgement

The first author was supported by a Krell Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. Research was also supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). This material is supported by U.S. Department of Energy grants numbered DE-SC0003959, DE-SC0004938, and DE-FC02-06-ER25786. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

Matrix multiplication on multidimensional torus networks

Edgar Solomonik and James Demmel

Division of Computer Science
University of California at Berkeley, CA, USA
solomon@cs.berkeley.edu, demmel@cs.berkeley.edu

Abstract. Blocked matrix multiplication algorithms such as Cannon's algorithm and SUMMA have a 2-dimensional communication structure. We introduce a generalized 'Split-Dimensional' version of Cannon's algorithm (SD-Cannon) with higher-dimensional and bidirectional communication structure. This algorithm is useful for higher-dimensional torus interconnects that can achieve more injection bandwidth than single-link bandwidth. On a bidirectional torus network of dimension d, SD-Cannon can lower the algorithmic bandwidth cost by a factor of up to d. With rectangular collectives, SUMMA also achieves the lower bandwidth cost but has a higher latency cost. We use Charm++ virtualization to efficiently map SD-Cannon on unbalanced and odd-dimensional torus network partitions. Our performance study on Blue Gene/P demonstrates that an MPI version of SD-Cannon can exploit multiple communication links and improve performance.

1 Introduction

Torus interconnects can scale to hundreds of thousands of nodes because they achieve good bisection bandwidth while maintaining bounded router degree on each node. Additionally, many scientific simulation and physically structured codes can be mapped to exploit locality on torus networks. In particular, 3-dimensional (3D) tori have been widely deployed in networks (e.g. IBM Blue Gene/L, Blue Gene/P, and the Cray XT series). The newest generation of high-end supercomputer networks is beginning to move to higher dimensionality (e.g. IBM Blue Gene/Q is 5D [4], K computer is 6D [14]). This transition is natural since the minimal-cost (bisection bandwidth with respect to number of pins) topology for a network of 100,000 nodes is 3D, while for 1,000,000 nodes it is 5D or 6D [5]. Higher-dimensional interconnects motivate the design of algorithms that can use such networks efficiently. In this paper, we adapt a classical matrix multiplication algorithm to exploit full injection bandwidth on a torus network of any dimension.

Cannon's algorithm [3] is a parallel algorithm for matrix multiplication ($C = A \cdot B$) on a square ($\sqrt{p}$-by-$\sqrt{p}$) processor grid. After staggering the initial matrix layout, Cannon's algorithm performs $\sqrt{p}$ shifts of A and B along the two dimensions of the processor grid. The algorithm can be done in-place and all communication is efficiently expressed in the form of near-neighbor data passes. Given n-by-n matrices, each processor must send $O(n^2/\sqrt{p})$ words of data in $O(\sqrt{p})$ messages along each dimension. The amount of words and messages sent by each node in Cannon's algorithm is asymptotically optimal [2] assuming minimal memory usage. However, since each node sends messages to nearest neighbors in 2 dimensions, at most 2 network links can be saturated per node. However, a d-dimensional bidirectional torus network has $2d$ outgoing links per node that can be utilized.

It is known that a different algorithm, SUMMA [12], can utilize all $2d$ links and send a minimal number of words. For matrix multiplication of n-by-n matrices, SUMMA sends $O(n^2/\sqrt{p})$ data in the form of n outer-products, which can be pipelined or blocked. Each update requires a broadcast along a row or column of processors. If a higher-dimensional torus is flattened into each row and column of the mapping, rectangular collective algorithms [13, 6, 10] can utilize all dimensions of the network. Rectangular algorithms subdivide and pipeline the messages into edge-disjoint spanning trees formed by traversing the network in different dimensional orders. However, SUMMA typically sends more messages since it does $O(\sqrt{p})$ broadcasts, rather than the $O(\sqrt{p})$ near-neighbor sends in Cannon's algorithm.

Cannon's algorithm does not employ communication collectives, so it cannot utilize rectangular collectives. We design a generalization of Cannon's algorithm, Split-Dimensional Cannon's algorithm (SD-Cannon), that explicitly sends data in all dimensions of the network at once. This algorithm does not need topology-aware collectives and retains all the positive features of the classical Cannon's algorithm. However, like Cannon's algorithm, SD-Cannon is difficult to generalize to non-square processor grids. We get around this challenge by using a virtualization framework, Charm++ [9]. Our performance results on Blue Gene/P (Intrepid, located at Argonne National Lab) demonstrate that SD-Cannon outperforms Cannon's algorithm and can match the performance of SUMMA with rectangular collectives. The virtualized version of Cannon's algorithm does not incur a high overhead but our Charm++ implementation is unable to saturate all networks links at once.

The rest of the paper is structured as follows, Section 2 introduces the SD-Cannon algorithm, Section 3 gives a cost analysis of the SD-Cannon algorithm, Section 4 studies the performance of MPI and Charm++ implementations of SD-Cannon, and Section 5 concludes.

2 Algorithm

Matrix multiplication computes $C = A \cdot B$ where A is m-by-k, B is k-by-n, and C is m-by-n. Our algorithms target the case where $m \approx n \approx k$. Other optimizations or different algorithms (e.g. 1D blocking) may be worth considering when the matrices are rectangular but are out of the scope of this paper. The algorithms are designed for a l-ary d-cube bidirectional torus network (a d dimensional network of l^d processors). The algorithms require that the torus network is of even dimension ($d = 2i$ for $i \in \{1, 2, ...\}$). Virtualization will be used to extend our approach to odd-dimensional networks and rectangular processor grids.

2.1 Matrix layout

A matrix is a 2D array of data. To spread this data in a load balanced fashion, we must embed the 2D array in the higher-dimensional torus network. We do not consider algorithms that replicate data (e.g. 3D matrix multiplication [1]). However, the extension is natural, given the replication approach presented in [11].

Any l-ary d-cube, where d is a multiple of 2, can be embedded onto a square 2D grid. Each of two dimensions in this square grid is of length $l^{d/2}$ and is formed by folding a different $d/2$ of the d dimensions. For simplicity we fold the odd $d/2$ dimensions into one of the square grid dimensions and the $d/2$ even dimensions into the other square grid dimension. The algorithms below will assume the initial matrix layout follows this ordering. In actuality, the ordering is irrelevant since a l-ary d-cube network is invariant to dimensional permutations. We define a dimensional embedding

$$G^{dD \to 2D}(I^d) = \left(\sum_{i=0}^{d/2-1} l^i I^d(2i), \sum_{i=0}^{d/2-1} l^i I^d(2i+1) \right)$$

2.2 Cannon's algorithm

We describe Cannon's algorithm on a 2D $\sqrt{P}$-by-$\sqrt{P}$ grid (Π^{2D}) (Algorithm 1).

Once we embed the dD grid onto a 2D grid, we can run Cannon's algorithm with the matrix distribution according to the ordered 2D processor grid. However, in this embedded network, Cannon's algorithm will only utilize $1/d$ of the links, since two messages are sent at a time by each processor and there are $2d$ links per node.

2.3 Split-dimensional Cannon's algorithm

We can formulate another version of Cannon's algorithm by using more dimensional shifts. A shift can be performed with a single message sent over a single link from each processor to the next. Since the shifts will be done along dimensions of the l-ary d-cube network, $2d$ links will be available. Algorithm 2

Algorithm 1: Cannon(A, B, C, n, m, k, P, Π^{2D})

Input: $m \times k$ matrix A, $k \times n$ matrix B distributed so that $\Pi^{2D}(i,j)$ owns $\frac{m}{\sqrt{P}} \times \frac{k}{\sqrt{P}}$ sub-matrix $A[i,j]$ and $\frac{k}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ sub-matrix $B[i,j]$, for each i,j

Output: square $m \times n$ matrix $C = A \cdot B$ distributed so that $\Pi^{2D}(i,j)$ owns $\frac{m}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ block sub-matrix $C[i,j]$, for each i,j

```
//In parallel with all processors
for all $i, j \in [0, \sqrt{P})$ do
    for $t = 1$ to $\sqrt{P} - 1$ do
        if $t \leq i$ then
            $A[i,j] \leftarrow A[i, ((j+1) \mod \sqrt{P})]$          /* stagger A */
        end
        if $t \leq j$ then
            $B[i,j] \leftarrow B[((i+1) \mod \sqrt{P}), j]$          /* stagger B */
        end
    end
    $C[i,j] := A[i,j] \cdot B[i,j]$
    for $t = 1$ to $\sqrt{P} - 1$ do
        $A[i,j] \leftarrow A[i, ((j-1) \mod \sqrt{P})]$          /* shift A rightwards */
        $B[i,j] \leftarrow B[((i-1) \mod \sqrt{P}), j]$          /* shift B downwards */
        $C[i,j] := C[i,j] + A[i,j] \cdot B[i,j]$
    end
end
```

performs this dimensional shift. Split-dimensional (SD) Cannon's algorithm will use exclusively this shift for communication. In fact, all shifts operate on a static message size. Therefore, communication cost can be calculated by counting shifts. The algorithm achieves complete utilization on any l-ary d-cube network during the shift stage. We specify the algorithm for a bidirectional network as those are much more common. However, the algorithms can be trivially simplified to the unidirectional case.

Algorithm 2: Shift$<$ **dim**, **dir** $>$(l, M, P, Π^{dD}, I^d)

Input: $\Pi^{dD}(I^d)$ owns sub-matrix M.

```
$S^d \leftarrow I^d$
if $dir = +1$ then
    $S^d(\text{dim}) = (S^d(\text{dim}) + 1) \mod l$
end
if $dir = -1$ then
    $S^d(\text{dim}) = (S^d(\text{dim}) - 1) \mod l$
end
Send $M$ to $\Pi^{dD}(S^d)$.          /* $\Pi^{dD}(I^d)$ sends to $\Pi^{dD}(S^d)$ */
```

Algorithm 3 describes how the stagger step is done inside the SD-Cannon algorithm. A different shift is done on each sub-panel of A and B concurrently. These calls should be done asynchronously and ideally can fully overlap. One interpretation of this stagger algorithm is that sub-panels of each matrix are being staggered recursively along $d/2$ disjoint dimensional orders.

Algorithm 4 is a recursive routine that loops over each dimension performing shifts on sub-panels of A and B. The order of the shifts is permuted for each sub-panel. Each sub-panel is multiplied via a recursive application of Cannon's algorithm over a given dimensional ordering.

Algorithm 3: SD-Stagger(A, B, n_b, m_b, k_b, l, P, Π^{dD},I^d)

```
Input: $\Pi^{dD}(I^d)$ owns $m_b \times k_b$ sub-matrix $A$ and $k_b \times n_b$ sub-matrix $B$.

Split $A = [A_0, A_1, \ldots, A_d]$ where each $A_h$ is $m_b \times k_b/d$.
Split $B^T = [B_0^T, B_1^T, \ldots, B_d^T]$ where each $B_h$ is $k_b/d \times n_b$.
//At each level, apply index shift
for $level \in [0, d/2-1]$ do
    //To stagger must shift up to $l-1$ times
    for $t = 1$ to $l-1$ do
        for all $dh \in [0, d/2-1]$ do
            $h \leftarrow (dh + \text{level}) \mod (d/2)$ //Shift the ordering
            if $t \leq I^d(2*h+1)$ then
                Shift< $(2*h), +1$ >($l$, $A_h$, $P$, $\Pi^{dD}$, $I^d$)
            end
            if $t \leq l - I^d(2*h+1)$ then
                Shift< $(2*h), -1$ >($l$, $A_{h+d/2}$, $P$, $\Pi^{dD}$, $I^d$)
            end
            if $t \leq I^d(2*h)$ then
                Shift< $(2*h+1), +1$ >($l$, $B_h$, $P$, $\Pi^{dD}$, $I^d$)
            end
            if $t \leq l - I^d(2*h)$ then
                Shift< $(2*h+1), -1$ >($l$, $B_{h+d/2}$, $P$, $\Pi^{dD}$, $I^d$)
            end
        end
    end
end
```

Algorithm 4: SD-Contract< level >(A, B, C, n_b, m_b, k_b, l, P, Π^{dD}, I^d)

```
Input: $\Pi^{dD}(I^d)$ owns $m_b \times k_b$ sub-matrix $A$ and $k_b \times n_b$ sub-matrix $B$.

Split $A = [A_0, A_1, \ldots, A_d]$ where each $A_h$ is $m_b \times k_b/d$.
Split $B^T = [B_0^T, B_1^T, \ldots, B_d^T]$ where each $B_h$ is $k_b/d \times n_b$.
//Shift and contract $l$ times
for $t = 0$ to $l-1$ do
    if $level = d-1$ then
        $C \leftarrow C + A \cdot B$
    else
        SD-Contract< level + 1 >($A$, $B$, $C$, $\frac{n}{\sqrt{P}}$, $\frac{m}{\sqrt{P}}$, $\frac{k}{\sqrt{P}}$, $l$, $P$, $\Pi^{dD}$, $I^d$)
    end
    for all $dh \in [0, d/2-1]$ do
        $h \leftarrow (dh + \text{level}) \mod (d/2)$                    /* Shift the ordering */
        Shift< $(2*h), +1$ >($l$, $A_h$, $P$, $\Pi^{dD}$, $I^d$)
        Shift< $(2*h+1), +1$ >($l$, $B_h$, $P$, $\Pi^{dD}$, $I^d$)
        Shift< $(2*h), -1$ >($l$, $A_{h+d/2}$, $P$, $\Pi^{dD}$, $I^d$)
        Shift< $(2*h+1), -1$ >($l$, $B_{h+d/2}$, $P$, $\Pi^{dD}$, $I^d$)
    end
end
```

Algorithm 5: SD-Cannon(A, B, C, n, m, k, l, P, Π^{dD}, $G^{dD\to 2D}$)

Input: $m \times k$ matrix A, $k \times n$ matrix B distributed so that Π_I^{dD} owns $\frac{m}{\sqrt{P}} \times \frac{k}{\sqrt{P}}$ sub-matrix $A[G^{dD\to 2D}(I)]$ and $\frac{k}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ sub-matrix $B[G^{dD\to 2D}(I)]$

Output: square $m \times n$ matrix $C = A \cdot B$ distributed so that Π_I^{dD} owns $\frac{m}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ block sub-matrix $C[G^{dD\to 2D}(I)]$

```
//In parallel with all processors
for all $I^d \in \{0, 1, \ldots, l-1\}^d$ do
    $(i, j) \leftarrow G^{dD\to 2D}(I^d)$
    SD-Stagger($A[i,j]$, $B[i,j]$, $\frac{n}{\sqrt{P}}$, $\frac{m}{\sqrt{P}}$, $\frac{k}{\sqrt{P}}$, $l$, $P$, $\Pi^{dD}$, $I^d$)
    SD-Contract$< 0 >$($A[i,j]$, $B[i,j]$, $C[i,j]$, $\frac{n}{\sqrt{P}}$, $\frac{m}{\sqrt{P}}$, $\frac{k}{\sqrt{P}}$, $l$, $P$, $\Pi^{dD}$, $I^d$)
end
```

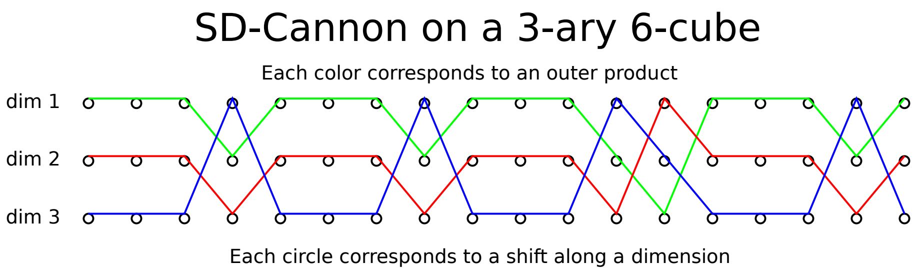

SD-Cannon on a 3-ary 6-cube
Each color corresponds to an outer product
dim 1
dim 2
dim 3
Each circle corresponds to a shift along a dimension

Fig. 1. Snapshot of network usage in SD-Cannon. A and B use a disjoint set of 3 network dimensions in the same fashion, so it suffices to pay attention to 3.

Figure 1 demonstrates how different network dimensions of a 3-ary 6-cube are used by SD-Cannon. A and B get shifted along 3 of 6 dimensions, so Figure 1 records usage along 3 dimensions (corresponding to one of A or B). Each outer product (corresponding to a color) is shifted (each shift corresponds to a circle) along a different dimension at any given step.

Note that the local multiplication call is the same as in Cannon's algorithm. The granularity of the sequential work does not decrease in the SD-Cannon algorithm but only changes its ordering. This is a virtue of splitting into outer-products that accumulate to the same buffer.

The control flow of SD-Cannon is described in Algorithm 5. The algorithm can be elegantly expressed with one-sided communication since the sends should be asynchronous (puts). Our MPI SD-Cannon code uses one-sided Put operations and is compact (a few hundred lines of C).

3 Analysis

We can analyze the bandwidth cost of these algorithms by the embedding of the algorithm onto the physical l-ary d-cube network. The bandwidth cost of the algorithm is proportional to the number of shifts along the critical path.

In traditional Cannon's algorithm we shift $2\sqrt{P}$ blocks along the critical path ($\sqrt{P}$ times for stagger and $\sqrt{P}$ times for contraction) of size mk/P and nk/P. Given the ordering of the embedding, we can always find a link which has to communicate $mk/\sqrt{P}$ values and a link that has to communicate $nk/\sqrt{P}$ values.[1]

[1] This analysis does not consider the wrap around pass. Some messages might have to go multiple hops, but this problem is relieved by interleaving the ordering of the processor grid.

Therefore the link bandwidth is

$$C_{\text{Cannon}} = \frac{2 \cdot \max(m, n) \cdot k}{\sqrt{P}}.$$

In the bidirectional, split-dimensional algorithm, all shifts in the communication inner loops (in Algorithm 3 and Algorithm 4) can be done simultaneously (so long as the network router can achieve full injection bandwidth). So the communication cost is simply proportional to the number of inner loops, which, for staggering (Algorithm 3) is $N_T = l \cdot d/2$. For the recursive contraction step (Algorithm 4), the number of these shift stages is $N_S = \sum_{i=1}^{d/2} l^i \leq 2\sqrt{p}$. If the network is bidirectional, at each shift stage we send asynchronous messages of sizes $\frac{mk}{d \cdot P}$ and $\frac{kn}{d \cdot P}$ values. Ignoring the lower-order stagger term in SD-Cannon we have a cost of

$$C_{\text{SD-Cannon}} = \frac{2 \cdot \max(m, n) \cdot k}{d \cdot \sqrt{P}}.$$

So the bandwidth cost of SD-Cannon, $C_{\text{SD-Cannon}}$, is d times lower than that of Cannon's algorithm, C_{Cannon}. The bandwidth cost of SUMMA is a factor of 2 smaller,

$$C_{\text{SUMMA}} = \frac{\max(m, n) \cdot k}{d \cdot \sqrt{P}}.$$

The latency overhead incurred by these algorithms will differ depending on the topology and collective algorithms for SUMMA. The SD-Cannon algorithm sends more messages than Cannon's algorithm, but into different links, so it incurs more sequential and DMA overhead, but no extra network latency overhead. However, both Cannon's algorithm and SD-Cannon will have a lower latency cost than SUMMA on a typical network. In each step of SUMMA, broadcasts are done along each dimension of the processor grid. So, on a torus network, a message must travel $l \cdot d/2$ hops at each step, rather than 1 hop as in SD-Cannon . The Blue Gene/P machine provides efficient broadcast collectives that work at a fine granularity and incur little latency overhead [6]. However, on a machine without this type of topology-aware collectives, SD-Cannon would have a strong advantage, as messages would need to travel fewer hops.

4 Results

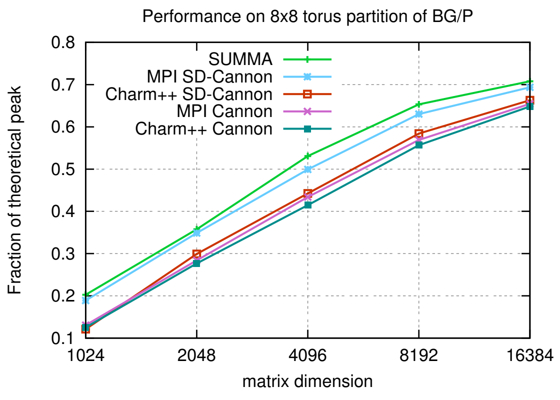
We implemented version of SD-Cannon in MPI [7] and Charm++ [9]. Both versions work on matrices of any shape and size, but we only benchmark square matrices. Both versions assume the virtual decomposition is a k-ary n-cube. In MPI, the process grid is a k-ary n-cube, while in Charm++ we get a k-ary n-cube of chare objects. We use Charm++ to explicitly map the objects onto any unbalanced process grid we define at run time. While we explicitly define the mapping function to fold the chare array onto a smaller processor grid, the Charm++ run-time system manages how the sequential work and messaging get scheduled.

The MPI version uses MPI Put operations for communication and barriers for synchronization. The Charm++ version uses the underlying run-time system for messaging between chares, and is dynamically scheduled (no explicit synchronization).

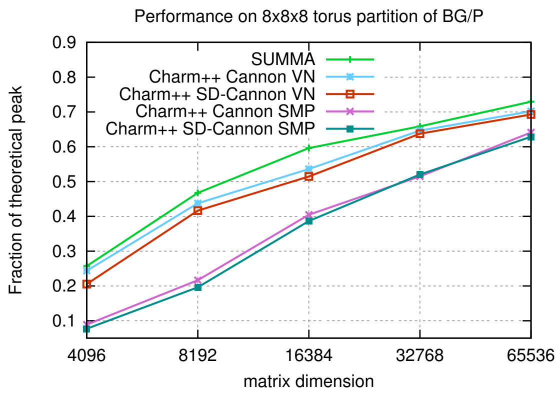
We benchmarked our implementations on a Blue Gene/P (BG/P) [8] machine located at Argonne National Laboratory (Intrepid). We chose BG/P as our target platform because it uses few cores per node (four 850 MHz PowerPC processors) and relies heavily on its interconnect (a bidirectional 3D torus with 375 MB/sec of achievable bandwidth per link).

Since the BG/P network only has three dimensions, the benefit of SD-Cannon is limited to trying to exploit the backwards links and the third dimensional links. The MPI version of our code is limited to even-dimensional tori, so it could only exploit 4 of 6 links. We study relative performance of the MPI version of SD-Cannon on an 8-by-8 torus partition of BG/P. [2]

[2] The partitions allocated by the BG/P scheduler are only toroidal if they have 512 or more nodes. So, we allocated a 512 node partition and worked on the bottom 64 node slice.



(a) Matrix multiplication on a 2D torus partition



(b) Matrix multiplication on a 3D torus partition

Fig. 2. Performance of SD-Cannon on BG/P

Figure 2(a) details the performance of MPI versions of SUMMA, Cannon's algorithm, and SD-Cannon on an 8x8 node 2D torus partition. SD-Cannon improves upon Cannon's algorithm as it can utilize the backwards as well as the forwards links simultaneously. The one-dimensional rectangular broadcasts used by SUMMA achieve the same effect. We see that the performance of SD-Cannon is higher than Cannon's algorithm (up to 1.5x) and slightly worse than SUMMA. The performance difference between SUMMA and SD-Cannon is due to the extra cost of the initial stagger, which SUMMA does not need. The Charm++ versions were executed with 1 chare per node. In this case, we see that Charm++ has a small overhead and we see a small benefit from bidirectionality.

Figure 2(b) shows the performance on a 3D 512 node partition. Since this partition is odd-dimensional, we cannot efficiently map the MPI version of Cannon or SD-Cannon. We execute the Charm++ codes in two modes, one with 1 process per node and 8 chares per process (SMP), and one with 4 processes per node and 2 chares per process (VN). Using multiple processes per node improves the performance of the Charm++ codes, because it is more efficient to perform a separate multiplication on each core, rather than execute each multiplication in sequence across all cores. While the VN-mode version performs almost as well as SUMMA, neither version benefits from multidimensional shifts. Our Charm++ implemenations use two-sided communication, while the MPI version uses one-sided. It is likely that a Charm++ implementation with one-sided sends would successfully exploit all of the links.

5 Conclusion

SD-Cannon is an improvement on top of Cannon's algorithm. While Cannon's algorithm has some nice properties, SUMMA has seen more wide-spread adaption. In this paper, we demonstrate how SD-Cannon can get closer to the performance of the SUMMA algorithm, and how virtualization can be used to map SD-Cannon and Cannon's algorithm efficiently. On the Blue Gene hardware it still does not make sense to use SD-Cannon over SUMMA, but SD-Cannon has advantages that could prove to be faster than SUMMA on other hardware.

More generally, our work demonstrates how algorithmic design can couple with topology-aware mapping and virtualization. These techniques are already important on modern supercomputers with 3D interconnects as demonstrated by our performance results. As the scale and dimensionality of high performance networks grow, topology-aware mapping and communication-avoidance are becoming pivotal to scalable algorithm design.

6 Acknowledgements

The first author was supported by a Krell Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. Research was also supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). This material is supported by U.S. Department of Energy grants numbered DE-SC0003959, DE-SC0004938, and DE-FC02-06-ER25786. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

1. Agarwal, R.C., Balle, S.M., Gustavson, F.G., Joshi, M., Palkar, P.: A three-dimensional approach to parallel matrix multiplication. IBM J. Res. Dev. 39, 575–582 (September 1995)
2. Ballard, G., Demmel, J., Holtz, O., Schwartz, O.: Minimizing communication in linear algebra. To appear in SIAM J. Mat. Anal. Appl (2011)
3. Cannon, L.E.: A cellular computer to implement the Kalman filter algorithm. Ph.D. thesis, Bozeman, MT, USA (1969)
4. Chen, D., Eisley, N.A., Heidelberger, P., Senger, R.M., Sugawara, Y., Kumar, S., Salapura, V., Satterfield, D.L., Steinmacher-Burow, B., Parker, J.J.: The ibm blue gene/q interconnection network and message unit. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 26:1–26:10. SC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2063384.2063419
5. Dally, W.: Performance analysis of k-ary n-cube interconnection networks. Computers, IEEE Transactions on 39(6), 775 –785 (jun 1990)
6. Faraj, A., Kumar, S., Smith, B., Mamidala, A., Gunnels, J.: MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations. In: High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on (2009)
7. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: portable parallel programming with the message-passing interface. MIT Press, Cambridge, MA, USA (1994)
8. IBM Journal of Research and Development staff: Overview of the IBM Blue Gene/P project. IBM J. Res. Dev. 52, 199–220 (January 2008)
9. Kale, L.V., Krishnan, S.: Charm++: a portable concurrent object oriented system based on c++. In: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. pp. 91–108. OOPSLA '93, ACM, New York, NY, USA (1993), http://doi.acm.org/10.1145/165854.165874
10. Solomonik, E., Bhatele, A., Demmel, J.: Improving communication performance in dense linear algebra via topology aware collectives. In: Supercomputing, Seattle, WA, USA (Nov 2011)
11. Solomonik, E., Demmel, J.: Communication-optimal 2.5D matrix multiplication and LU factorization algorithms. In: Lecture Notes in Computer Science, Euro-Par, Bordeaux, France (Aug 2011)
12. Van De Geijn, R.A., Watts, J.: SUMMA: scalable universal matrix multiplication algorithm. Concurrency: Practice and Experience 9(4), 255–274 (1997)
13. Watts, J., Van De Geijn, R.A.: A pipelined broadcast for multidimensional meshes. Parallel Processing Letters 5, 281–292 (1995)
14. Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M., Watanabe, T.: The k computer: Japanese next-generation supercomputer development project. In: Low Power Electronics and Design (ISLPED) 2011 International Symposium on. pp. 371 –372 (aug 2011)