Analyzing the Performance of a Sparse Matrix Vector Multiply for Extreme Scale Computers

Amanda Bienz, Jon Calhoun, Luke Olson, Marc Snir, and William D. Gropp
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave.
Urbana, Illinois 61801, USA
{bienz2, jccalho2, lukeo, snir, wgropp}@illinois.edu

ABSTRACT

As high-performance computing systems continue to progress towards extreme scale, the scalability of applications becomes critical. The scalability of an algorithm is dependent on interconnect properties, such as latency and bandwidth, and is often limited by network contention. Sparse matrix-vector multiplication (SpMV) is fundamental to a large class of HPC applications. We investigate the performance and scalability of SpMV routines in the widely used software packages PETSc, Hypre, and Trilinos. Through the use of an asynchronous multiplication, we show an improvement in scalability and performance of the SpMV operation when applied to various matrices.

1. INTRODUCTION

The large amount of parallelism in high-performance computing systems yields the potential for growing levels of computation. This increase in parallelism, combined with machine design constraints, has created new hurdles, e.g. network contention, for algorithms to overcome in order to be scalable and efficient.

The sparse matrix-vector multiply (SpMV) is fundamental to a large class of HPC applications. The communication required for each SpMV is dependent on the matrix being multiplied. Matrix reorderings have shown potential to improve performance, but can incur substantial cost [2]. Leveraging node-level parallelism and architectural features [3] improves performance with respect to the computation, but at extreme scale communication can become the dominating factor inhibiting performance.

Improved performance can be achieved on extreme scale systems by utilizing asynchronous communication. Communicating asynchronously consists of initializing communication, performing local operations, and then completing communication. This strategy permits the overlap of communication and computation, yielding the potential to greatly reduce idle-time, and hence, the total time required by the algorithm.

```
Input: $A_l$, $x_l$, $b_l$
for $i = 0$ to $\#sends$ do
    MPI_Isend...
    MPI_Irecv...
mult(diag, $x_{local}$, $b$)
MPI_Waitall(...)
mult(offd, $x_{distant}$, $b$)
```

(a) Original

```
Input: $A_l$, $x_l$, $b_l$
for $i = 0$ to $\#sends$ do
    MPI_Isend...
    MPI_Irecv...
mult(diag, $x_{local}$, $b$)
while recvs_outstanding
    MPI_Waitsome...
    if complete
        mult(offd, $x_{proc[i]}$, $b$)
        recvs_outstanding -= 1
```

(b) Asynchronous

Figure 4: SpMV algorithms.

In this paper we present: (1) An asynchronous SpMV algorithm; and (2) A comparison of time required in variations of the original SpMVs with those present in HYPRE, PETSc, and Trilinos.

2. BACKGROUND

Sparse parallel matrices are often partitioned row-wise, as shown in Figure 1, with each process holding a contiguous portion of the rows of the matrix, and the equivalent rows of the vector. The local rows of the matrix are further partitioned into two sets of columns; a diagonal block and an off-diagonal block. The diagonal block corresponds to columns whose associated vector components are local. Therefore, the portion of each SpMV that consists of multiplying entries from the diagonal block requires no communication. The off-diagonal blocks, however, corresponds to vector components that are stored on distant processors.

3. ASYNCHRONOUS SPMV

To obtain the vector components for the off-diagonal blocks, linear algebra software often issues standard MPI calls, such as `MPI_Isend` and `MPI_Irecv`. While this communication is occurring, the local diagonal block of the matrix is often multiplied. However, once this local computation finishes, the process waits for all receives to complete before proceeding with the off-diagonal portion of the SpMV, as described in the algorithm, Figure 4a. Thus one late message delays all of the computation.

Our optimization, Figure 4b, waits for any communication to complete through the use of `MPI_Waitsome`. The portion of the off-diagonal block corresponding to the these receives is multiplied as soon as this communication completes. Processing the off-diagonal SpMV on a per-receive basis, yields

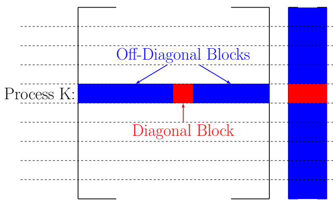

Figure 1: Matrix and vector partitioning for a parallel SpMV.

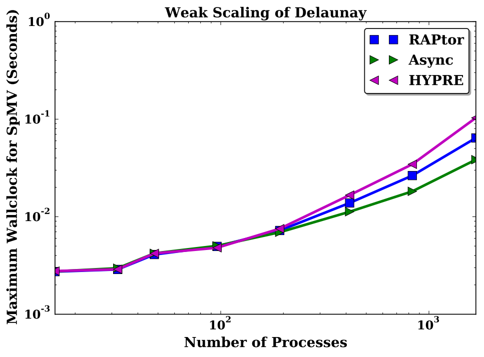

Figure 2: Maximal time for SpMV with Delaunay Matrices on **Vulcan**.

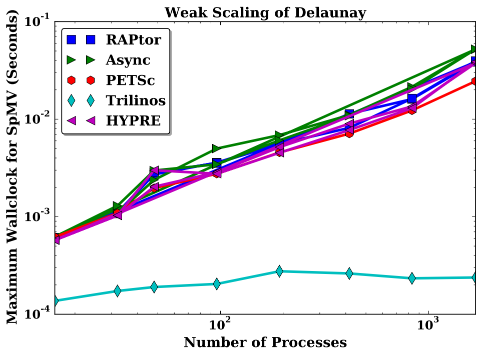

Figure 3: Maximal time for SpMV with Delaunay Matrices on **Blue Waters**.

a pipelined off-diagonal SpMV as communication overlaps with computation.

4. EXPERIMENTAL RESULTS

We compare the time required to perform a parallel SpMV from system installed HYPRE, PETSc, and Trilinois on Vulcan and Blue Waters to both our original synchronous algorithm, RAPtor, as well as its asynchronous counterpart (Async). These tests were performed with 16 processes per node using the University of Florida Sparse Matrix Collection [1] matrices: Delaunay from the DIMACS10 group with approximately 10,000 degrees-of-freedom per process. The network performance on Blue Waters and Vulcan differs significantly. In the context of a postal execution model, we measure $\alpha = 1.812e^{-6}$ and $\beta = 1.845e^{-9}$ for Blue Waters and $\alpha = 4.319e^{-6}$ and $\beta = 4.274e^{-9}$ on Vulcan. Parallel SpMVs rely on point to point messages, and the difference in α affects the viability of our asynchronous approach.

4.1 Vulcan

Figure 2 shows the maximal time required for a SpMV for the Delaunay matrices, which form an arrowhead, yielding a load imbalance among the processes. At small numbers of processes we see no difference between the SpMVs. However, as we weak scale we see Async out preforming HYPRE by 2.7x for 1,680 processes. We do not test PETSc or Trilinios on Vulcan due to them not being system installed for full optimizations.

4.2 Blue Waters

When run on Blue Waters, as shown in Figure 3, we see the performance parity between our synchronous and asynchronous approach along with all the packages except Trilinios. This is due to the per message latency being 4x faster than Vulcan. This limits the extent to which communication and computation can overlap. At much larger core counts, network contention will become a larger factor, delaying messages and leading to a greater ability to overlap communication and computation.

5. CONCLUSIONS

Overlapping communication and computation through a pipelined off-diagonal SpMV can greatly improve the performance of a communication dominated SpMV. However, communication costs greatly vary across machines, allowing asynchronous SpMV to greatly improve runtime on Vulcan while decreasing performance when the same problem is run on Blue Waters. Furthermore, matrices with the majority of non-zero entries located near the main diagonal require little communication, yielding no benefit from the pipelined approach. The optimal SpMV method is dependent on both matrix density as well as network speed. When network contention is high, our asynchronous becomes tractable. Our approach lends itself well to a task based parallelization, and suggests future study is needed to fully needed to understand its benefits and trade-offs.

Acknowledgment

This work was sponsored by the Air Force Office of Scientific Research under grant FA9550-12-1-0478. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI–0725070 and ACI–1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant Number DGE-1144245.

6. REFERENCES

[1] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[2] Torsten Hoefler and Marc Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 75–84, New York, NY, USA, 2011. ACM.

[3] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, pages 38:1–38:12, New York, NY, USA, 2007. ACM.