Communication Bounds for Heterogeneous Architectures

(Regular Submission)

Grey Ballard
UC Berkeley
ballard@cs.berkeley.edu

James Demmel
UC Berkeley
demmel@cs.berkeley.edu

Andrew Gearhart
UC Berkeley
agearh@cs.berkeley.edu

Abstract

As the gap between the cost of communication (i.e., data movement) and computation continues to grow, pursuing algorithms which minimize communication has become a critical research objective. Toward this end, we seek asymptotic communication lower bounds for general memory models and classes of algorithms. Recent work [2] has established lower bounds for a wide set of linear algebra algorithms on a sequential machine and on a parallel machine with identical processors. This work extends these previous bounds to a heterogeneous model in which processors access data and perform floating point operations at differing speeds. We also present algorithms which prove that the lower bounds are tight (i.e., attainable) in the cases of dense matrix-vector and matrix-matrix multiplication.

Keywords: Communication lower bounds, heterogeneity, dense linear algebra

1 Introduction

Traditionally, algorithms for dense linear algebra focus on minimizing floating point operations as the key to performance. However, as the gap between interprocessor communication and computation continues to grow, focus has shifted to the problem of reducing data movement between various levels of a memory hierarchy or between different nodes on a parallel machine. Thus, the idea of a "communication-avoiding" (CA) algorithm has become a concept of critical research importance. CA algorithms differ from traditional flop-minimized approaches as they asymptotically minimize communication, sometimes at the cost of extra flops. Solving a problem in a CA manner may involve a significant amount of algorithmic innovation (for examples, see [6, 8, 12]).

Toward the goal of developing CA algorithms (and ultimately producing efficient software), we seek asymptotic communication lower bounds for general memory models and certain classes of algorithms. Once an optimal algorithm for a memory model has been established, then lower-level tuning can obtain high performance on specific architectures.

Recent work [2] has established communication lower bounds for a large set of linear algebra algorithms on both sequential and parallel machines. However, the parallel model assumes homogeneity among processors. In this work, we propose a heterogeneous parallel machine model and show how the lower bounds of [2] can be extended to this more complicated model. These new bounds hold for the same wide set of algorithms as the prior work. Further, we argue that the new lower bounds are asymptotically tight by presenting algorithms that attain them (to within constant factors).

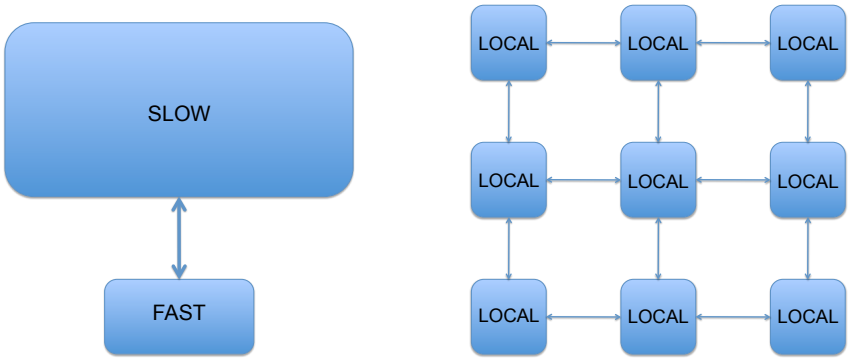
We differentiate between communication costs that are dominated by the initial reading of inputs and final writing of outputs of an algorithm and those that are dominated by the shuffling of data in and out of memory. The former case is common with $O(n^2)$ matrix-vector operations, while the latter encompasses $O(n^3)$ matrix-matrix operations and matrix factorizations where the data is too large to fit into fast memory at once.

The rest of this paper is organized as follows. We discuss prior communication lower bounds in Section 2. We present the heterogeneous model in Section 3 and establish lower bounds for the model in Section 4. In Sections 5 and 6 we present algorithms for matrix-vector multiplication and matrix-matrix multiplication, respectively, which attain the lower bounds. Section 7 concludes and discusses future research directions.

2 Prior Communication Lower Bounds

A communication lower bound for dense matrix multiplication was first proved for a sequential machine in the two-level memory model (shown in Figure 1(a)) by Hong and Kung [9]. Irony, Toledo, and Tiskin [10] extended the result for dense matrix multiplication to parallel machines where the data is distributed across the local memories of the processors (i.e., the distributed memory model, shown in Figure 1(b)). In this model, all of the processors and their local memory sizes are identical. The proof in [10] is based on a geometric inequality of Loomis and Whitney [11] which is used to bound the arithmetic intensity for matrix multiplication on a processor with a given fast memory size.

Recent work [2] has generalized the communication lower bound to many other linear algebra algorithms. The more general proof is also based on the Loomis-Whitney [11] inequality. This result holds for most of "direct" linear algebra algorithms including Basic Linear Algebra Subroutine (BLAS) [4] operations (e.g. matrix-vector multiplication, matrix multiplication, and triangular solve with one or multiple right hand sides) and computing LU, Cholesky, LDL^T, and QR decompositions, as well as many eigenvalue/SVD computations. The result holds whether the matrices are dense or sparse, and whether the machine fits the two-level or distributed memory model. If a processor does G floating point operations (flops) that satisfy



(a) Two-level sequential model (b) Distributed parallel model

Figure 1: Previous memory models

the requirements described in [2] and has M words of local memory (fast memory in the sequential case), then the total number of words W sent and received by the processor satisfies

$$W \geq \frac{G}{8\sqrt{M}} - M.$$

Also, as observed in [2], if the original input data does not reside in fast memory at the start of the algorithm and the final output data must be written out of fast memory at the end of the algorithm, then there is a trivial lower bound based on the combined number of input and output words. Since the negative term in the bound above accounts for the possibility of the fast memory being full of useful data at the start of the algorithm, we can ignore it under this assumption. Thus, a tighter lower bound may be expressed as

$$W \geq \max\left(I + O, \frac{G}{8\sqrt{M}}\right). \tag{1}$$

where I and O are the number of input and output words, respectively. Note that the second term in the maximum function degenerates to less than one when G is sufficiently small relative to the size of the local memory, which is the case for dense algorithms in which the input and output matrices fit in local memory and for some sparse problems in which the sparsity structure yields few flops. In the case of dense matrix multiplication of $n \times n$ matrices on a sequential machine with $n^2 \gg M$, inequality (1) reproduces $W = \Omega\left(\frac{n^3}{\sqrt{M}}\right)$ as in [9].

In the distributed memory model, words are packed into contiguous messages before being communicated to another processor. In the two-level memory model, words which are stored contiguously in slow memory can be read into fast memory (or written back to slow memory) in a single message. We represent the time cost of a single message between fast/local memory and slow/global memory as:

$$T_{\text{msg}} = \alpha + \beta w$$

where w is the number of words transferred. In order to model the latency costs (i.e., the cost of a message that is independent of message size), we are also interested in the number of messages each processor sends and receives.

Following [2], we obtain a lower bound on the number of messages L a processor sends and receives by dividing the lower bound on the number of words given in (1) by the fast/local memory size M, which is the size of the largest possible message. Thus, if a processor executes G flops as before, we have

$$L \geq \max\left(\frac{I + O}{M}, \frac{G}{8M^{3/2}}\right). \tag{2}$$

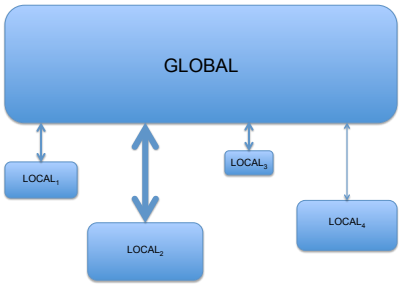


Figure 2: Heterogeneous memory model

3 A Model for Heterogeneous Architectures

To capture the non-uniform nature of a heterogenous computing environment, we model the machine to be a set of compute elements proc$_i$ ($1 \leq i \leq P$) with varying characteristics connected via independent links to a shared global memory. Figure 2 shows a graphical representation of the model. Each proc$_i$ can be defined very abstractly; for example, one compute element may be a single processor, GPU, or a shared-memory multiprocessor itself. Each proc$_i$ is associated with several element-specific parameters:

- β_i: Inverse bandwidth of proc$_i$'s communication link to global shared memory
- α_i: Latency of proc$_i$'s communication link to global shared memory
- M_i: Size of the local memory of proc$_i$
- γ_i: Floating point performance (flops per second) of proc$_i$ on data that resides in local memory

We assume that the initial data for the problem is stored within the machine's global memory. For the purposes of this work, global memory is assumed to be without size constraints. Note that unlike the distributed parallel model, the heterogeneous model presented here assumes that the initial problem lies within the global memory. Thus, the layout of data in global memory becomes a factor for performance: in order to read/write a set of words in one message, those words must be contiguously stored in global memory.

Note that when $i = 1$, this model reduces to the sequential two-level memory model shown in Figure 1(a). However, in the case that all the element-specific parameters are equivalent across compute elements (e.g., $\beta_i = \beta$ for $1 \leq i \leq P$), this heterogeneous model does not reduce to the parallel distributed-memory model shown in Figure 1(b) since it is based on a global shared memory.

A natural extension of this model is to apply it to real architectures in a hierarchical way. In the case that proc$_i$ represents multiple computational units (heterogeneous or not) with a shared memory, one could apply this model to the element individually to obtain a more accurate prediction of γ_i.

4 Lower Bounds for Heterogeneous Architectures

Suppose we run an algorithm which executes G flops on a heterogeneous machine, and suppose the algorithm assigns F_i flops to proc$_i$ for $1 \leq i \leq P$, such that $\sum F_i = G$. Then we can focus our attention on one compute element proc$_i$ and model the communication between the local memory of proc$_i$ and machine global memory as two levels of a sequential machine. In this way we obtain a lower bound on the number of words W_i transferred to/from proc$_i$ by applying inequality (1) and, similarly, a lower bound on the number of messages L_i by applying inequality (2). Although we can obtain separate lower bounds for each compute

element, the bounds apply only to a particular partitioning of the total flops. We would like a lower bound which applies to any assignment of the G flops to the different compute elements.

Toward this end, we broaden our focus from the individual communication costs of each compute element to the total parallel runtime. We begin by recalling the time cost of a single message in our model between proc$_i$'s local memory and global memory as $T_{\text{msg}_i} = \alpha_i + \beta_i w$, where w is the number of words transferred. From this, by ignoring idle time, we lower bound proc$_i$'s total runtime T_i by the sum of three terms:

$$T_i \geq \gamma_i F_i + \beta_i W_i + \alpha_i L_i.$$

This runtime model ignores any potential overlap of computation and communication, though we note that completely overlapping computation and communication will decrease the runtime by at most a factor of $2\times$.

In the heterogeneous model, the parallel runtime is determined by the last compute element to finish its computation. Thus, given a partition $\{F_i\}$ of the G flops (i.e., $\sum F_i = G$, we have

$$T(\{F_i\}) \geq \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i W_i + \alpha_i L_i$$

where F_i, W_i, and L_i are the number of flops executed, words communicated, and messages communicated, respectively, by proc$_i$ during the course of the algorithm. In order to obtain a more general lower bound, we can find the minimum over all possible partitions $\{F_i\}$ with $\sum F_i = G$, yielding

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i W_i + \alpha_i L_i. \tag{3}$$

Assuming that inequalities (1) and (2) hold, we can apply them to (3) to obtain our heterogeneous lower bound on parallel runtime:

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i \max\left\{ I_i + O_i, \frac{F_i}{8\sqrt{M_i}} \right\} + \alpha_i \max\left\{ \frac{I_i + O_i}{M_i}, \frac{F_i}{8M_i^{3/2}} \right\}. \tag{4}$$

In the following two subsections, we will identify two circumstances where this lower bound may be greatly simplified. In each case, the simplifications will suggest algorithms which can attain the lower bound to within constant factors. In this way, we will argue that lower bound given in (4) is asymptotically tight for the problems solved by these algorithms.

4.1 Input/Output Dominated Lower Bound

In this section, we focus on the lower bound based on original inputs and final outputs for each proc$_i$. That is, if we ignore the lower bound guaranteed by the result based on Loomis-Whitney, we obtain another valid lower bound which may be lower than the one in (4). This input/output dominated lower bound, given by

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i (I_i + O_i) + \alpha_i \left(\frac{I_i + O_i}{M_i} \right) \tag{5}$$

is valid for any algorithm where the original input and final output must reside in global memory. We may simplify this bound for an algorithm with a direct relationship between flops and input and output data.

For example, in the case of BLAS2 [4] operations like n-by-n dense-matrix-vector-multiplication, $G = O(n^2)$ and $I + O = O(n^2)$. Thus, the number of inputs and outputs in BLAS2 functions are related to the number of flops via some constant c. So, we can represent inequality (5) as

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i (c F_i) + \alpha_i \left(\frac{c F_i}{M_i} \right)$$

or

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \xi_i F_i,$$

where

$$\xi_i = \gamma_i + c\beta_i + \frac{c\alpha_i}{M_i} \tag{6}$$

and c is a constant determined by the specific number of words per flop for a given BLAS2 algorithm.

We can further simplify this min-max expression by solving the associated linear program. Observe that the minimum is attained when $\xi_i F_i$ is constant for $1 \leq i \leq P$ (i.e., the compute elements finish simultaneously), and we discover that a partition attaining the minimum satisfies

$$F_i = \frac{\frac{1}{\xi_i}}{\sum_j \frac{1}{\xi_j}} G \tag{7}$$

for $1 \leq i \leq P$. Thus, for BLAS2 operations, we obtain the partition-independent, input/output dominated lower bound

$$T \geq \max_{1 \leq i \leq P} \xi_i F_i = \frac{G}{\sum_j \frac{1}{\xi_j}}. \tag{8}$$

Again, inequality (8) may not be as tight a bound as (4) in general, but we will argue in Section 5 that it can be attained in the case of matrix-vector multiplication. This will imply that in that case, both bounds are equivalent and tight.

4.2 Loomis-Whitney Dominated Lower Bound

In this section, on the other hand, we focus on the lower bound based on Loomis-Whitney. This time, ignoring the lower bound guaranteed by having to read the original inputs and write the final outputs, we obtain another lower bound which may be lower than the one in (4). This Loomis-Whitney dominated lower bound is given by

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \gamma_i F_i + \beta_i \left(\frac{F_i}{8\sqrt{M_i}} \right) + \alpha_i \left(\frac{F_i}{8M_i^{3/2}} \right).$$

For example, this lower bound applies to BLAS3 [4] operations such as n-by-n dense-matrix-matrix-multiplication as well as most direct linear algebra algorithms where $G = O(n^3)$. We can rewrite this inequality as

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \delta_i F_i$$

where

$$\delta_i = \gamma_i + \frac{\beta_i}{8\sqrt{M_i}} + \frac{\alpha_i}{8M_i^{3/2}} \tag{9}$$

is a constant.

As before, we can simplify the min-max expression above by solving the associated linear program. This implies that the partitioning $\{F_i\}$ that attains the minimum satisfies

$$F_i = \frac{\frac{1}{\delta_i}}{\sum_j \frac{1}{\delta_j}} G \tag{10}$$

for $1 \leq i \leq P$. Thus, we obtain a partition-independent Loomis-Whitney dominated lower bound

$$T \geq \max_{1 \leq i \leq P} \delta_i F_i = \frac{G}{\sum_j \frac{1}{\delta_j}}. \tag{11}$$

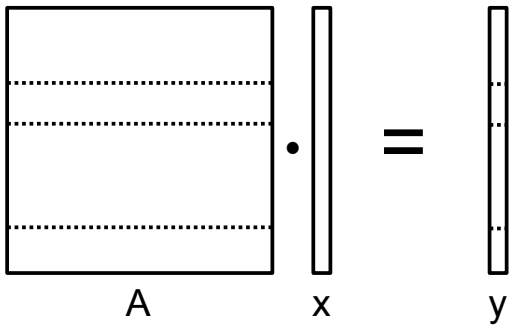


Figure 3: HGEMV splitting

While inequality (11) may not be as tight a bound as (4) in general, we will argue in Section 6 that it can be attained in the case of $O(n^3)$ matrix-matrix multiplication. This will imply that in that case, both bounds are equivalent and tight.

5 Matrix-Vector Multiplication

In this section we present an algorithm to compute square matrix-vector multiplication (GEMV) that attains the lower bound presented in Section 4.1. As a BLAS2 operation, square GEMV performs $2n^2$ flops upon $n^2 + 2n$ data. Thus, we do two flops per word of data transferred and the value of c in Equation (6) is $\frac{1}{2}$. With this definition of c, we can rewrite the lower bound of (8) as

$$T \geq \frac{G}{\sum_j \frac{1}{\xi_j}} = \xi_i F_i = \gamma_i F_i + \beta_i \left(\frac{1}{2} F_i\right) + \alpha_i \left(\frac{1}{2} \frac{F_i}{M_i}\right) \tag{12}$$

where F_i is defined as in equation (7) and we note that $\xi_i F_i$ is constant for $1 \leq i \leq P$.

An optimal algorithm for square GEMV on a heterogeneous machine is presented as Algorithm 1. This algorithm divides the work among the processors by partitioning the rows of the matrix A as shown in Figure 3. In this way, each processor computes a subset of the entries of the output vector and there are no write contentions (though each compute element must access the entire input vector).

Algorithm 1 Heterogeneous matrix-vector multiplication

Require: Matrix $A \in \mathbb{R}^{n \times n}$, stored in row-wise order, vector $x \in \mathbb{R}^n$,
1: Measure $a_i, \beta_j, \gamma_i, M_i$ for each $1 \leq i \leq P$ and set ξ_i according to equation (6) with $c = 1/2$
2: **for** $i = 1$ to P **do**
3: Set F_i according to equation (7) where $G = 2n^2$
4: Choose splitting rows r_i ($1 \leq i \leq P$, with $r_0 = 0$) in matrix A such that $r_i - r_{i-1} \approx \frac{F_i}{2n}$
5: **end for**
6: **for all** proc$_i$ ($1 \leq i \leq P$) **parallel do**
7: Compute $GEMV_i(A(r_{i-1} : r_i, :), x)$
8: **end for**

For simplicity, we assume the matrix A is stored in row-wise order so that each $GEMV_i$ is performed on a contiguous block of memory. Further, we assume that each $GEMV_i$ accesses matrix entries contiguously (along rows). This implies that if the input vector does not fit in fast memory ($n > M_i$), then each input vector entry must be read from slow memory for each row of the matrix. While a blocked algorithm (with

a blocked data structure) is more communication-efficient, the difference in the the number of words and messages transferred from slow memory is less than a constant factor ($2\times$).

To see that the parallel running time of this algorithm is within a constant factor of the lower bound given in equation (12), consider proc$_i$. It computes a matrix-vector product with a matrix of size $m_i \times n$ where $m_i = r_i - r_{i-1} \approx \frac{F_i}{2n}$. Thus, it performs $2m_in \approx F_i$ flops. Because the division of work is done by rows, even if $\frac{F_i}{2n}$ is not an integer, the processor is assigned no more than one row of extra work ($2n$ flops). Assuming each compute element is assigned at least one row of work, this implies that the number of flops done on the compute element is no more than $2\times$ that of the lower bound.

We now consider the communication costs. As mentioned above, if the flops are performed in row-wise order and the input vector x does not fit in fast memory, then two reads are required for each scalar multiplication. Since the cost of writing the output vector entries is a lower order term, the number of words transferred by proc$_i$ is also $2m_in$. Since $2m_in$ is within $2\times$ of F_i, the number of words transferred is within $4\times$ of the lower bound. Since we are accessing the matrix and input vector entries in contiguous order, we can read the data in blocks of size about $M_i/2$ (in order to fit both blocks in fast memory at the same time). Thus, the number of messages is about $\frac{4m_in}{M_i}$ which is within $8\times$ of the lower bound.

Since each term (flop cost, bandwidth cost, latency cost) of the running time of proc$_i$ is within a constant factor of the lower bound, the sum of the three terms is also within a constant factor. This argument holds for each compute element individually, so the maximum runtime over all compute elements (i.e., the parallel runtime) is within a constant factor of the lower bound given in equation (12). Thus, Algorithm 1 is asymptotically optimal within a factor of 8 of the lower bound. Note that by using a blocked data structure for the matrix and corresponding algorithm, one could obtain a runtime within $4\times$ of the lower bound.

6 Matrix Multiplication

In this section we present an algorithm to compute square matrix-matrix multiplication (GEMM) that attains the lower bound presented in Section 4.2. For comparison to the upper bound analysis, we re-write inequality (11) in terms of three summands. Letting F_i be defined as in equation (10), and noting that $\delta_i F_i$ is constant for $1 \leq i \leq P$, we have the lower bound

$$T \geq \frac{G}{\sum_j \frac{1}{\delta_j}} = \delta_i F_i = \gamma_i F_i + \beta_i \left(\frac{F_i}{8\sqrt{M_i}} \right) + \alpha_i \left(\frac{F_i}{8M_i^{3/2}} \right). \tag{13}$$

We will base the algorithm on the square recursive matrix multiplication algorithm (see [5] for example). In this algorithm, each of the matrices are divided into four $\frac{n}{2} \times \frac{n}{2}$ submatrices and the blocked multiplication of these submatrices yields eight subproblems of $2(n/2)^3$ flops each, which can be solved recursively. We assume that n is a power of two in this section.

We require that the $n \times n$ input matrices A and B are stored in a block-recursive format. The block-recursive format [1, 7, 13] (also known as the bit interleaved layout, space-filling curve storage, or Morton ordering format) stores each of the four $\frac{n}{2} \times \frac{n}{2}$ submatrices contiguously, and the elements of each submatrix are ordered so that the smaller submatrices are each stored contiguously, and so on recursively. In this way, every subproblem within square recursive GEMM will be associated with contiguous data.

At a high level, the algorithm assigns subproblems of various sizes to each processor in a manner consistent with the optimal flop distribution as suggested by the lower bound. It assigns as many subproblems at one level of recursion as possible before recursing to smaller subproblems. The flop assignments are represented as octal fractions in order to determine the number and size of subproblems to assign to each processor. Also, no subproblem is allocated such that the entire problem fits in the assigned processor's fast memory.

Algorithm 2 Heterogeneous matrix-matrix multiplication

Require: Matrices $A, B \in \mathbb{R}^{n \times n}$, stored in block-recursive order, n is a power of two

1: Measure $a_i, \beta_j, \gamma_i, M_i$ and set δ_i according to equation (9) for each $1 \leq i \leq P$
2: **for** $i = 1$ to P **do**
3: Set F_i according to equation (10) where $G = n^3$
4: Set k_i to be the largest integer such that $3(n/2^{k_i})^2 \geq M_i$
5: Convert F_i/G into octal and round[1] to k_i^{th} digit: $0.d_1^{(i)} d_2^{(i)} \cdots d_{k_i}^{(i)}$
6: **end for**
7: Initialize $S = \{A \cdot B\}$
8: **for** $j = 1$ to $\max k_i$ **do**
9: Subdivide all problems in S into 8 subproblems according to square recursive GEMM
10: Assign $d_j^{(i)}$ subproblems to proc_i and remove subproblems from S
11: **end for**
12: **for all** proc_i **parallel do**
13: Compute assigned subproblems using square recursive GEMM
14: **end for**

Ensure: Matrix $C = AB$, stored in block-recursive order

We will assume that for a given heterogeneous machine, the problem size is large enough such that the distribution of flops to compute elements according to equation (10) satisfies

$$F_i \geq (M_i/3)^{3/2} \tag{14}$$

for each $1 \leq i \leq P$. Note that on a sequential machine, this degenerates to $3n^2 \geq M$, where the matrix multiplication problem (two input matrices and one output matrix) is too large to fit entirely in fast memory. This assumption may be violated for a small problem on a heterogeneous machine where one compute element is relatively slow (i.e., large δ_i) but has a large fast memory (i.e., large M_i).

To see that the parallel runtime is within a constant factor of the lower bound given in equation (13), consider proc_i. As in Section 5, we will argue that each of the three terms contributing to proc_i's runtime are within constant factors of the corresponding terms in the lower bound.

Algorithm 2 does not assign exactly F_i flops to the proc_i. Instead, in line 5, F_i/G is rounded to a fraction with k_i octal digits.[1] Thus, the actual number of flops assigned is $U_i = \left(0.d_1^{(i)} d_2^{(i)} \cdots d_{k_i}^{(i)}\right)_8 \cdot G$, yielding

$$\frac{U_i}{G} - \frac{F_i}{G} \leq \frac{1}{8^{k_i}}.$$

Further, k_i is chosen in line 4 so that $3\left(\frac{n}{2^{k_i+1}}\right)^2 \leq M_i$ which implies $\frac{n^3}{8^{k_i}} \leq \left(\frac{4}{3}M_i\right)^{3/2}$. Since $G = n^3$, we have

$$U_i - F_i \leq \frac{n^3}{8^{k_i}} \leq \left(\frac{4}{3}M_i\right)^{3/2}.$$

By our assumption in inequality (14), we have

$$\frac{U_i - F_i}{F_i} \leq 8\,\frac{(M_i/3)^{3/2}}{F_i} \leq 8$$

[1]Rounding each of these fractions to a finite-digit octal representation such that the sum of octal fractions is exactly one is a nontrivial problem. For the purposes of this upper bound, we may assume that the rounding scheme always rounds up to the next k_i^{th} digit, in which case the sum will be greater than one.

and so the number of flops assigned to proc$_i$ in Algorithm 2 is within a constant factor of the flops given in the lower bound.

We now consider the communication costs for proc$_i$. By construction, the octal fraction representing the work assigned to proc$_i$ has no more than k_i digits. This implies that the smallest subproblem assigned to the compute element involves submatrices of size at least $n/2^{k_i} \times n/2^{k_i}$. Since $3(n/2^{k_i})^2 \geq M_i$, the smallest subproblem is too large to fit into fast memory. In this case, from [3], the number of words transferred by the square recursive GEMM on a sequential machine for each subproblem is $O(\#\text{flops}/\sqrt{M_i})$, and with a block recursive data structure, the number of messages is $O(\#\text{flops}/M_i^{3/2})$.[2] Thus, the total number of words transferred between proc$_i$ and slow memory is $O(U_i/\sqrt{M_i})$, and the number of messages transferred is $O(U_i/M_i^{3/2})$. Since U_i is within a constant factor of F_i, the number of words and messages transferred is within a constant factor of the lower bound.

We also note that for matrix multiplication, all subproblems are independent (ignoring the $O(n^2)$ work to sum the results of pairs of subproblems), so there is no idle time on processors due to data dependencies. Thus, the running time of each compute element is the sum of the three terms of arithmetic and communication costs. Since each of these terms is within a constant factor of the lower bound for each compute element, the maximum runtime over all compute elements is no more than a constant factor larger than the lower bound given in inequality (13). Thus, Algorithm 2 is optimal.

7 Future Work and Conclusions

In this work, we have extended existing communication lower bounds for simple memory models to a shared-memory heterogeneous model. The bounds hold for a wide class of algorithms in dense and sparse direct linear algebra including BLAS operations and matrix factorizations. Further, we have shown that the bounds are tight for dense matrix-vector and matrix-matrix multiplication by presenting algorithms that attain the lower bounds within constant factors.

While the runtimes of the algorithms in Sections 5 and 6 are asymptotically optimal, we would like to show empirically that the flop distributions suggested by the algorithms are efficient on real heterogeneous machines. The most common simple heterogeneous environment is a CPU paired with a graphics processing unit (GPU). Given accurate measurement of system parameters, we have preliminary results that show that in the case of matrix-vector multiplication our runtime model accurately represents measured performance. We hope to obtain similar results for the case of matrix-matrix multiplication.

Future work will be targeted in two directions: extending communication bounds to more complicated theoretical models and developing new linear algebra algorithms that attain the bounds for the model in this paper. In the first case, we would like to consider bounds on heterogeneous machines that include multiprocessors as compute elements. For example, such a hierarchical model would allow for representation of individual GPU cores in a CPU/GPU system. We also plan to explore bounding communication on distributed heterogeneous machines without a global memory, i.e. machines where each proc$_i$ possesses multiple communication links.

Futher algorithm work will explore ways to implement problems such as triangular solves, Cholesky, LU, and QR optimally within the heterogeneous model. While GEMV and GEMM allow for many independent computations and flexibility in flop distribution, these other algorithms include dependencies which make them harder to map to arbitrary heterogeneous machines.

We also believe we can extend these results to Strassen and other fast matrix multiplication algorithms. Given a lower bound for Strassen in the two-level memory model, the same arguments can be made to

[2] The analysis of recursive GEMM in [3] is for a more general algorithm which handles rectangular matrices. In the case of square matrices, the algorithm reduces to square recursive GEMM.

extend the lower bounds to the heterogeneous model. With slight modifications, we believe Algorithm 2 will provide a matching upper bound.

Acknowledgment

This research was supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).

References

[1] M. Bader, R. Franz, S. Günther, and A. Heinecke. Hardware-oriented implementation of cache oblivious matrix operations based on space-filling curves. In *Proceedings of the 7th international conference on parallel processing and applied mathematics*, PPAM '07, pages 628–638, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing Communication in Linear Algebra. Submitted. Available from http://arxiv.org/abs/0905.2485, 2009.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing*, 32(6):3495–3523, 2010.

[4] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subroutines (BLAS). *ACM Trans. Math. Soft.*, 28(2), June 2002.

[5] R. Blumofe, M. Frigo, C. Joerg, C. Leiserson, and K. Randall. Dag-consistent distributed shared memory. In *IPPS '96: Proceedings of the 10th international parallel processing symposium*, pages 132–141, 1996.

[6] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Implementing communication-optimal parallel and sequential QR and LU factorizations. Submitted to SIAM. J. Sci. Comp., 2008.

[7] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):pp. 3–45, 2004.

[8] L. Grigori, J. Demmel, and H. Xiang. Communication avoiding Gaussian elimination. In *Proceedings of the 2008 ACM/IEEE conference on supercomputing*, SC '08, pages 29:1–29:12, Piscataway, NJ, USA, 2008. IEEE Press.

[9] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on theory of computing*, pages 326–333, New York, NY, USA, 1981. ACM.

[10] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[11] L. H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the AMS*, 55:961–962, 1949.

[12] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the 2009 ACM/IEEE conference on supercomputing*, SC '09, pages 36:1–36:12, New York, NY, USA, 2009. ACM.

[13] D. Wise. Ahnentafel indexing into Morton-ordered arrays, or matrix locality for free. In Arndt Bode, Thomas Ludwig, Wolfgang Karl, and Roland Wismller, editors, *Euro-Par 2000 Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 774–783. Springer Berlin / Heidelberg, 2000.