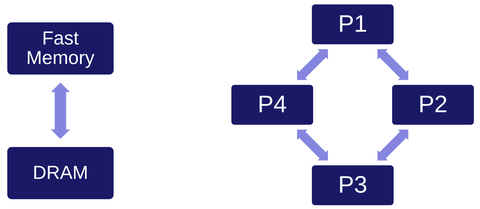
A Parallel Communication-Avoiding Biconjugate Gradient Algorithm

Erin Carson and Nick Knight

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

Motivation: The Cost of Communication

- The cost of an algorithm has two primary components: computation (floating point operations) and communication (movement of data to and from memory)
- Communication is comprised of two measurements: **bandwidth** (total number of words moved) and **latency** (the number of messages in which those words are sent)



Sequential Case **Parallel Case**

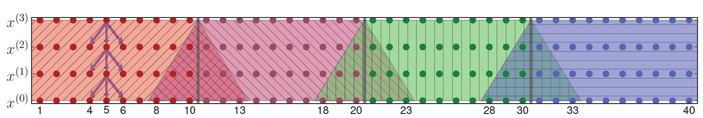
- Because flop rate is increasing at a faster rate than memory bandwidth or inverse memory latency, **computations are communication bound**
- **Motivation: We can improve the performance of iterative methods if we rearrange the algorithm to avoid communication.**

The Matrix Powers Kernel

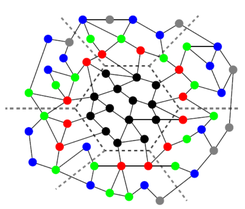
The matrix powers kernel takes a sparse matrix A, a dense vector x, and scalars $\lambda_1,\ldots,\lambda_k$ and outputs the vectors

$$(A-\lambda_1 I)x,(A-\lambda_2 I)(A-\lambda_1 I)x,\ldots,(A-\lambda_k I)(A-\lambda_{k-1} I)\ldots(A-\lambda_1 I)x$$

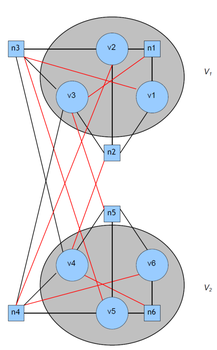
Whereas this operation would normally compute these vectors by performing k SpMV operations, the matrix powers kernel has been designed to compute all k vectors while **only reading the matrix *A* once**. This is accomplished by determining dependencies at the beginning of the computation, and ensuring that each partition has all required data to calculate its entries in the result vectors before doing any work. A method based on **hypergraph partitioning** is used to minimize dependencies, and thus minimizes redundant work. This **minimizes communication** both in terms of reading from slow memory and minimizing communication between threads.



Parallel algorithm for a tridiagonal matrix (n = 40, p = 4, k = 3). Vertices, representing elements of x (and rows of A), are colored by their affinities. Overlapping regions indicate redundant work. [MHDY09].



Representation of dependencies in a general graph with 7 partitions. In order to compute values for black vertices, we also need red vertices for k = 1, red and green for k = 2, etc. [MHDY09].



Hypergraph partitioning based on k-level column nets. Given a partition, the total communication volume can be found by determining the cut cost of the graph between vertices in each partition. Black edges represent dependencies for k = 1, and red and black edges represent dependencies for k = 2.

The CA-BiCG Algorithm

The Biconjugate gradient algorithm is a Krylov subspace method for the solution of linear systems that are not necessarily symmetric or positive definite. We rewrite the standard BiCG algorithm to make use of the Matrix Powers Kernel,

[A, x] → [x, Ax, …, A^kx],

which avoids communication by eliminating the k SpMVs in the inner loop. We can additionally use the matrix powers kernel and residual vectors to form a Gram matrix, which can then be used to eliminate dot products.

Algorithm 5: CA-BiCG

Input: $Ax = b$: $n \times n$ linear system
Input: $x_1, \hat{x}_1$: initial guess
Output: Approximate solution x_{sk+j} *of* $Ax = b$

$x_0 = 0,\ r_0 = 0,\ R_{-1} = 0_{n,s},\ r_1 = b - Ax_1$
$\hat{x}_0 = 0,\ \hat{r}_0 = 0,\ \hat{R}_{-1} = 0_{n,s},\ \hat{r}_1 = b - A^T\hat{x}_1$
for k $= 0,1,2,\ldots$ *, until convergence* ***do***
  $v_{sk+1} = r_{sk+1}$
  $\hat{v}_{sk+1} = \hat{r}_{sk+1}$
  Use matrix powers kernel to compute $\underline{V}_k$ *and* $\underline{\hat{V}}_k$
  $G_k = \begin{bmatrix} \hat{R}_{k-1}^T R_{k-1} & \hat{R}_{k-1}^T \underline{V}_k \\ \hat{V}_k^T R_{k-1} & \hat{V}_k^T \underline{V}_k \end{bmatrix}$
  Use Equation 1 to compute recurrence coefficients d_{sk} *and* $\hat{d}_{sk}$
  Use Equation 2 to compute recurrence coefficients g_{sk} *and* $\hat{g}_{sk}$
  for j $= 1,2,\ldots,s$ ***do***
    Use Equation 1 to compute recurrence coefficients $d_{sk+j}, \hat{d}_{sk+j}$ *such that* $Ar_{sk+j} = [R_{k-1}, \underline{V}_k]d_{sk+j}, A\hat{r}_{sk+j} = [\hat{R}_{k-1}, \underline{\hat{V}}_k]\hat{d}_{sk+j}$
    Use Equation 2 to compute recurrence coefficients $g_{sk+j}, \hat{g}_{sk+j}$ *such that* $r_{sk+j} = [R_{k-1}, \underline{V}_k]g_{sk+j},\ \hat{r}_{sk+j} = [\hat{R}_{k-1}, \underline{\hat{V}}_k]\hat{g}_{sk+j}$
    $w = [R_{k-1}, \underline{V}_k]d_{sk+j}$
    $\hat{w} = [\hat{R}_{k-1}, \underline{\hat{V}}_k]\hat{d}_{sk+j}$
    Compute $\langle \hat{r}_{sk+j},\ r_{sk+j} \rangle = \hat{g}_{sk+j}^T G_k g_{sk+j}$
    Compute $\langle Ar_{sk+j},\ \hat{r}_{sk+j} \rangle = \hat{g}_{sk+j}^T G_k d_{sk+j}$
    $\gamma_{sk+j} = \frac{\langle \hat{r}_{sk+j},\ r_{sk+j} \rangle}{\langle w,\ \hat{r}_{sk+j} \rangle}$
    if $sk + j = 1$ ***then***
      $\rho_{sk+j} = 1$
    else
      $\rho_{sk+j} = \left(1 - \frac{\gamma_{sk+j}}{\gamma_{sk+j-1}} \frac{\langle \hat{r}_{sk+j},\ r_{sk+j} \rangle}{\langle \hat{r}_{sk+j-1},\ r_{sk+j-1} \rangle} \frac{1}{\rho_{sk+j-1}}\right)^{-1}$
    end if
    $x_{sk+j+1} = \rho_{sk+j}(x_{sk+j} + \gamma_{sk+j} r_{sk+j}) + (1 - \rho_{sk+j})x_{sk+j-1}$
    $\hat{x}_{sk+j+1} = \rho_{sk+j}(\hat{x}_{sk+j} + \gamma_{sk+j} \hat{r}_{sk+j}) + (1 - \rho_{sk+j})\hat{x}_{sk+j-1}$
    $r_{sk+j+1} = \rho_{sk+j}(r_{sk+j} - \gamma_{sk+j} w_{sk+j}) + (1 - \rho_{sk+j})r_{sk+j-1}$
    $\hat{r}_{sk+j+1} = \rho_{sk+j}(\hat{r}_{sk+j} - \gamma_{sk+j} \hat{w}_{sk+j}) + (1 - \rho_{sk+j})\hat{r}_{sk+j-1}$
  end for
end for

$$d_{sk+j} = \begin{cases} 0_{2s+1,1} & k = 0, \\ [0_{1,s-1}, 1, 0_{1,s+1}]^T & k > 0, j = 0 \\ [0_{1,s}, \underline{B}_k(1,1), \underline{B}_k(2,1), 0_{1,s-1}]^T & \\ \rho_{sk+j-1}d_{sk+j-1} + (1-\rho_{sk+j-1})d_{sk+j-2} - & \\ \quad \rho_{sk+j-1}\gamma_{sk+j-1}\begin{bmatrix} T_{k-1}d_{sk+j-1}(1:s) \\ -(\rho_{sk}\gamma_{sk})^{-1}d_{sk+j-1}(s)e_1 + \underline{B}_k d_{sk+j-1}(s+1:2s) \end{bmatrix} & j = 1 \end{cases}$$

$$g_{sk+j} = \begin{cases} 0_{2s+1,1} & k = 0, j = 0 \\ [0_{1,s-1}, 1, 0_{1,s+1}]^T & k > 0, j = 0 \\ [0_{1,s}, 1, 0_{1,s}]^T & j = 1 \\ \rho_{sk+j-1}g_{sk+j-1} + (1-\rho_{sk+j-1})g_{sk+j-2} & \\ \quad - \rho_{sk+j-1}\gamma_{sk+j-1}d_{sk+j} & j = 2{:}s \end{cases}$$

$$\hat{d}_{sk+j} = \begin{cases} 0_{2s+1,1} & k = 0, \\ [0_{1,s-1}, 1, 0_{1,s+1}]^T & k > 0, j = 0 \\ [0_{1,s}, \underline{\hat{B}}_k(1,1), \underline{\hat{B}}_k(2,1), 0_{1,s-1}]^T & \\ \rho_{sk+j-1}\hat{d}_{sk+j-1} + (1-\rho_{sk+j-1})\hat{d}_{sk+j-2} - & \\ \quad \rho_{sk+j-1}\gamma_{sk+j-1}\begin{bmatrix} T_{k-1}\hat{d}_{sk+j-1}(1:s) \\ -(\rho_{sk}\gamma_{sk})^{-1}\hat{d}_{sk+j-1}(s)e_1 + \underline{\hat{B}}_k \hat{d}_{sk+j-1}(s+1:2s) \end{bmatrix} & j = 1 \end{cases}$$

$$\hat{g}_{sk+j} = \begin{cases} 0_{2s+1,1} & k = 0, j = 0 \\ [0_{1,s-1}, 1, 0_{1,s+1}]^T & k > 0, j = 0 \\ [0_{1,s}, 1, 0_{1,s}]^T & j = 1 \\ \rho_{sk+j-1}\hat{g}_{sk+j-1} + (1-\rho_{sk+j-1})\hat{g}_{sk+j-2} & \\ \quad - \rho_{sk+j-1}\gamma_{sk+j-1}\hat{d}_{sk+j} & j = 2{:}s \end{cases}$$

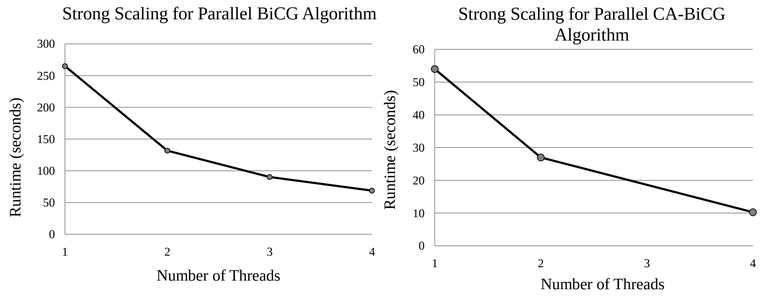
Shared Memory Parallel Implementation

We used pthreads to write a shared memory parallel implementation of CA-BiCG (and BiCG). Each thread "owns" rows of matrices and entries of vectors, done with a naïve partitioning, which all reside in shared memory. Although iterations in the algorithm must be performed sequentially, the operations within the loop can be easily parallelized. We parallelized the algorithm in the following ways:

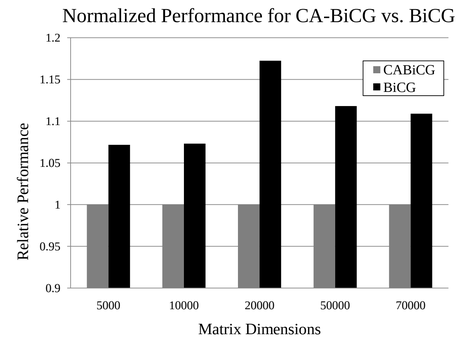
- The matrix powers kernel computes the k basis vectors in parallel
- Each thread computes its entries of the search direction vector (small SpMV), according the to rows of the matrix it owns
- Dot products are performed in parallel
- Each thread computes updated values for the entries it owns in the result vectors and residual vectors

Performance Results

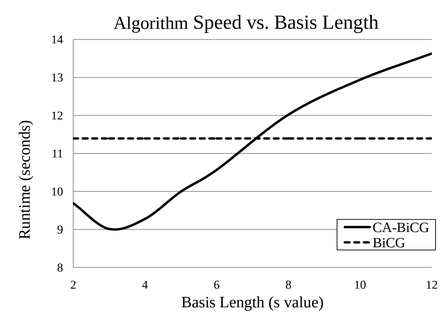
We implemented both a shared memory parallel version of standard BiCG and a shared memory parallel version of CA-BiCG. All tests were run on the Intel Xeon X5550 (Nehalem) architecture, which is dual socket, quad core. The first experiment illustrates strong scaling of both algorithms for up to 4 threads.



Our second experiment involved comparing the standard BiCG method to our communication-avoiding method. We gathered timing data, also on the Nehalem architecture, for tridiagonal matrices of increasing size. We see speedups up to 17%.



The matrix powers kernel avoids an additional communication for each increase of s by 1, but also incurs additional computational cost, since the surface-to-volume ratio increases with the fill-in. After some value of s, this extra work will exceed the savings of avoiding communication, and become worse than the traditional parallel BiCG algorithm. To explore this behavior, we ran CA-BiCG on a 20000x20000 tridiagonal matrix, with varying s values. The minimum runtime occurs at s=3, although performance increases are noted between s=2 through 7. We expect these features to shift to the right in environments where communication is more expensive.



Future Work

- Create Matrix Powers Kernel implementation that will allow us to compute the Krylov subspace for both A and A^T , so we can test on nonsymmetric matrices
- Further experiments on convergence properties
  - Implementation of other bases (Newton, Chebyshev)
- Explore opportunities for co-tuning: Can we achieve further speedups?
- Extend this algorithm to CA-BiCGStab in order to improve stability
- Exploit more opportunities for parallelism (computing recurrence coefficients, eliminating dot products by use of Gram Matrix, etc.)

The Parallel Computing Laboratory