Topology-Aware Mappings for Large-Scale Eigenvalue Problems

Hasan Metin Aktulga[1], Chao Yang[1], Esmond G. Ng[1],
Pieter Maris[2], and James P. Vary[2]

[1] Lawrence Berkeley National Laboratory, Berkeley CA 94720, USA
[2] Iowa State University, Ames IA 50011, USA

Abstract. Obtaining highly accurate predictions for properties of light atomic nuclei using the Configuration Interaction (CI) approach requires computing the lowest eigenvalues and associated eigenvectors of a large many-body nuclear Hamiltonian matrix, $\hat{H}$. Since $\hat{H}$ is a large sparse matrix, a parallel iterative eigensolver designed for multi-core clusters is used. Due to the extremely large size of $\hat{H}$, thousands of compute nodes are required. Communication overhead may hinder the scalability of the eigensolver at such scales. In this paper, we discuss how to reduce such overhead. In particular, we quantitatively show that topology-aware mapping of computational tasks to physical processors on large-scale multi-core clusters may have a significant impact on efficiency. For typical large-scale eigenvalue calculations, we obtain up to a factor of 2.5 improvement in overall performance by using a topology-aware mapping.

1 Introduction

High fidelity scientific simulations are nowadays carried out on multi-core machines that consist of thousands or tens of thousands of nodes. Hopper, a Cray XE6 machine at the National Energy Research Scientific Center (NERSC), has 6,384 nodes and 24 cores per node. Kraken, a Cray XT5 platform at the National Institute for Computational Sciences (NICS), has 9,408 nodes and 12 cores per node. These nodes and cores are connected by a sophisticated communication network that has a limited bandwidth. The bandwidth per flop ratio (BPF) of these machines has exhibited a declining trend. For the successive Cray models XT4, XT5 and XE6, the BPF ratios are 1.36, 0.23 and 0.10 bytes per flop, respectively [2,3]. On exascale platforms, the BPF ratio is anticipated to be much lower. This trend puts enormous pressure on scientific software developers to devise new algorithms and implementations that have minimal communication overhead. One way to achieve this goal is to develop algorithms that have a lower communication volume and fewer number of messages. Another strategy, which we will focus on in this paper, is to change the way processes are mapped to physical processors so that the load on the interconnection network, which is characterized by a number of metrics such as the average link dilation, traffic and congestion, can be reduced. With this strategy, it is possible to reduce the communication overhead in large-scale parallel computations [1].

C. Kaklamanis et al. (Eds.): Euro-Par 2012, LNCS 7484, pp. 830–842, 2012.

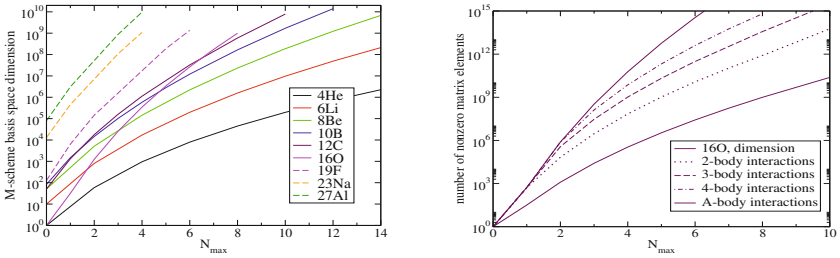

Fig. 1. The dimension and the number of non-zero matrix elements of the nuclear Hamiltonian with respect to $N_{\max}$ and the number of particles A

In this paper, we show that changes in task-to-processor mapping can have a drastic effect on the performance of large sparse eigenvalue calculations for predicting nuclear structures. We explain the observed effects quantitatively by reporting various load metrics associated with different mappings. Our results confirm the need to seek a topology-aware mapping to reduce communication overhead for large-scale parallel computations.

2 Eigensolver for the CI Approach

The key problem to be solved in nuclear structure calculations is the nuclear many-body Schrödinger's equation $H\psi = E\psi$, where ψ is a many-body wavefunction and H is a nuclear many-body Hamiltonian. One way to solve the problem is to expand ψ in terms of Slater determinants of single-particle basis functions that satisfy a number of contraints. A particular choice of single-particle basis suitable for nuclear structure calculation is the harmonic oscillator basis. The representation of H under this basis expansion, which is often referred to as the configuration interaction (CI) approach, is a sparse symmetric matrix $\hat{H}$. The dimension of $\hat{H}$, which we denote by n, is defined by the number of Slater determinants used in the expansion, which is in turn determined by the number of nucleons A and a constraint on the harmonic oscillator quanta, which is often denoted by $N_{\max}$. Higher $N_{\max}$ values yield more accurate results for the same nucleus, but at the expense of an exponential growth in problem size, see Fig. 1.

Due to the large dimension and the sparsity of $\hat{H}$, an iterative algorithm such as the Lanczos algorithm is preferred to solve the eigenvalue problem described above. The basic steps of the Lanczos algorithm are outlined in Alg. 1. The computational cost of the algorithm is generally dominated by the first two steps within the *while* loop: (i) multiplication of the sparse matrix $\hat{H}$ with the most recent Lanczos vector v, (ii) orthogonalization of the new vector w with respect to previous Lanczos vectors stored in V (a renormalization step may also be desirable). In this section, we discuss how these two tasks are decomposed into subtasks and how the subtasks are mapped to processing units for achieving a load balanced parallel implementation of Alg. 1.

Algorithm 1. The basic steps of the Lanczos Algorithm

Input: $\hat{H}$, v_0;
Output: ψ and E such that $\|\hat{H}\psi - \psi E\|_F$ is small.
$v \leftarrow v_0/\|v_0\|$;
$V \leftarrow (v)$;
while not converged **do**
  $w \leftarrow \hat{H}v$;
  $w \leftarrow w - VV^T w$;
  $v \leftarrow w/\|w\|$;
  $V \leftarrow (V, v)$;
  $T \leftarrow V^T \hat{H} V$;
  Diagonalize T to obtain (U, E);
  Check convergence;
end while
$\psi = VU$;

2.1 Sparse Matrix Vector Multiplication (SpMV)

On a distributed memory machine, the SpMV multiplication $w \leftarrow \hat{H}v$ can be carried out in parallel by partitioning the rows and columns of $\hat{H}$ and distributing the nonzero elements of $\hat{H}$ among different processing units. This is the strategy taken by the state-of-the-art nuclear CI calculation software package MFDn (Many-body Fermion Dynamics for nuclear physics) [6,7]. An even distribution of non-zero matrix elements over all processors is crucial for optimizing the use of available memory and achieving good load-balance. This is accomplished in MFDn by an appropriate matrix reordering through row/column permutation.

Since $\hat{H}$ is symmetric, we store only its lower triangular part to reduce memory usage. Consequently, it is natural to organize processing units into an $n_{\rm d} \times n_{\rm d}$ triangular grid over the $\hat{H}$ matrix, as shown in Fig. 2. Each processing unit, which stores the (i, j)th portion of the sparse matrix $\hat{H}$, can be labeled by its row and column positions on the grid. The total number of processing units n_p in this triangular grid is $n_{\rm d}(n_{\rm d}+1)/2$, where $n_{\rm d}$ is also the number of processing units along the diagonal.

A simple way to perform the SpMV multiplication in parallel is to partition the vector v by rows into $\{v_i\}$, where $i = 1, 2, ..., n_{\rm d}$. in a way that is conformal with the column partitioning of $\hat{H}$ as shown in Fig. 2. Row and column communication groups, labeled by the communicator $C_{\rm row}$ and $C_{\rm col}$ respectively, are set up to allow v_i to be broadcast among processing units that lie on the ith row or column of the triangular grid. If we denote the submatrix of $\hat{H}$ assigned to the (i, j)th processing unit by $\hat{H}_{i,j}$, each processing unit performs two SpMVs of the form $w_i = \hat{H}_{i,j} v_j$ and $w_j = \hat{H}^T_{i,j} v_i$. Two reductions are required (one along the row communication groups and one along the column communication groups) to merge local products w_i and w_j to form the global result vector w.

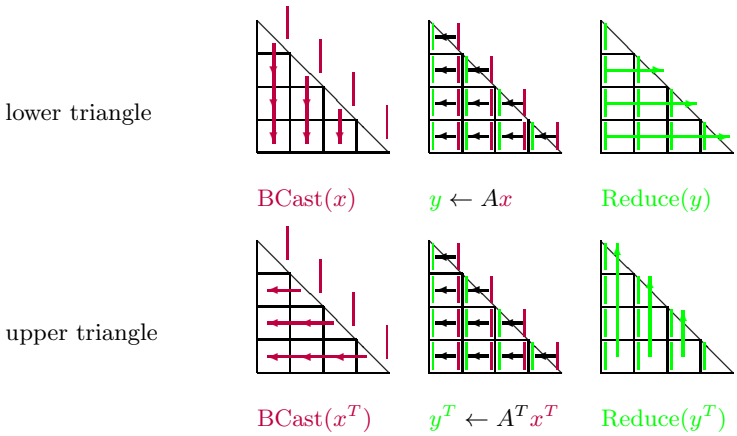

Fig. 2. A visual illustration of the communication pattern for the distributed SpMV procedure, as implemented in MFDn

If n is the dimension of $\hat{H}$, the length of each distributed Lanczos vector is roughly n/n_{d}. The communication volume associated with the broadcast and reduction operations required for SpMV multiplication is $\mathrm{O}(nn_{\mathrm{d}})$ along the C_{col} and C_{row} groups. The dependence of the communication volume on the number of diagonal processors n_{d} suggests that a multi-threaded (hybrid MPI/OpenMP parallel) implementation running on the same number of cores as a pure MPI implementation would have less communication overhead. To be precise, a multi-threaded implementation with t threads would generate $\sqrt{t}$ times less communication volume compared to the pure MPI version. Because in the hybrid MPI/OpenMP implementation with t-way thread parallelism, n_{d} decreases by a factor of $\sqrt{t}$, while n does not change.

2.2 Basis Orthogonalization

In MFDn, the orthogonalization of a new basis vector w against a previously constructed orthonormal basis contained in the matrix V is parallelized on a $n_{\mathrm{d}} \times (n_{\mathrm{d}}+1)/2$ 2D grid reconfigured from the $n_{\mathrm{d}} \times n_{\mathrm{d}}$ triangular grid as shown in Fig. 3. To maximize the amount of parallelism while minimizing communication volume, we distribute V by using a hierarchical 1D distribution scheme. A basis vector v is first divided into n_{d} subvectors v_j, $j = 1, 2, ..., n_{\mathrm{d}}$, each associated with the jth row group of the 2D grid. Each subvector v_j is then further partitioned into $(n_{\mathrm{d}} + 1)/2$ parts and distributed among the processing units in the same row group of the 2D grid.

After the SpMV multiplication $w = \hat{H}v$ is completed, a scattering operation is required to distribute w_j conforming to the distribution of the v_j subvectors among the jth row group of the 2D grid for $j = 1, 2, ...n_{\mathrm{d}}$. Once w has been distributed among all processing units, an all-reduce operation is required to complete the orthogonalization $w - VV^T w$. Communication volume associated with this all-reduce operation is typically small, when V does not contain too

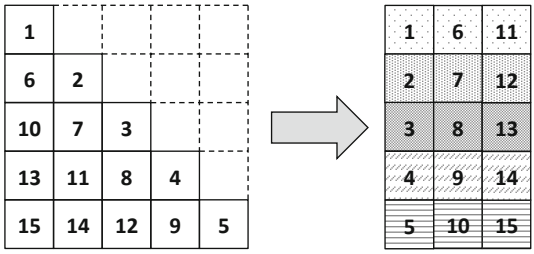

Fig. 3. Reconfiguring a $n_d \times n_d$ lower triangular processing grid (left) into a $n_d \times (n_d + 1)/2$ rectangular grid (right) for basis orthogonalization

many basis vectors. Finally, since the next SpMV in our iterative eigensolver requires the new basis vector v_i to be available on all processing units within the same column (or row) communication group, a gathering operation is required. The scattering and gathering operations involve a communication volume of $\mathcal{O}(n)$ only. Therefore, when the basis vectors are partitioned hierarchically in 1D, the total communication volume of the basis orthogonalization part is $\mathcal{O}(n)$, which is considerably smaller than that of the SpMV part.

3 Estimating the Communication Overhead

While the communication volume often provides a good estimate of the efficiency of a parallel program, the actual performance of the program will depend on other factors. The mapping between computational tasks and physical processing units has a strong influence, too. Therefore we need additional metrics that can capture the effect of process topology on the actual performance of the parallel program.

3.1 Network Load Model

We use a simplified version of the framework suggested by Hoefler and Snir [1]. A communication graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ is a directed graph, where $\mathcal{V}_\mathcal{G}$ is the set of processes and an edge $e = \{u, v\} \in \mathcal{E}_\mathcal{G}$ denotes a message sent from process $u \in \mathcal{V}_\mathcal{G}$ to $v \in \mathcal{V}_\mathcal{G}$. We define two communication graphs, $\mathcal{G}_{\text{col}}$ and $\mathcal{G}_{\text{row}}$, which are associated with the column and row communication groups of the triangular grid, respectively.

Similarly, the physical interconnection network is represented as a directed, weighted graph $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H}, c_\mathcal{H})$, where $\mathcal{V}_\mathcal{H}$ is the set of compute nodes, $\mathcal{E}_\mathcal{H}$ is the set of links between these nodes, and $c_\mathcal{H}(e)$ corresponds to the bandwidth of a link $e = \{u, v\}$. We assume that messages are routed between nodes using the shortest path. We denote such a path by $p(u, v)$, the set of links connecting nodes u and v. Since there is usually more than one such shortest path, $\mathcal{P}(u, v)$ is used to refer to the set of all shortest paths between u and v. We assume that each path in $\mathcal{P}(u, v)$ is used with equal probability for sending a message.

We also assume static routing, *i.e.*, messages are not redirected when congestion is detected on certain parts of the network.

Under the model described above, we define $\Gamma : \mathcal{V}_\mathcal{G} \rightarrow \mathcal{V}_\mathcal{H}$ as a function that maps the vertices of a communication graph to the physical nodes in the network graph. Three quality measures can be defined for a mapping Γ: (average) dilation $\mathcal{D}(\Gamma)$, (average) traffic $\mathcal{T}(\Gamma)$ and (maximum) congestion $\mathcal{X}(\Gamma)$. These definitions are similar to those given by Hoefler and Snir [1], but slightly simpler due to the assumptions stated above. $\mathcal{D}(\Gamma)$ is the average number of links traveled by a message:

$$\mathcal{D}(\Gamma) = \frac{\sum_{\{u,v\}\in\mathcal{E}_\mathcal{G}} |p(\Gamma(u), \Gamma(v))|}{|\mathcal{E}_\mathcal{G}|} \tag{1}$$

Dilation is a measure of the total communication work that needs to be performed by the interconnection network. As dilation increases, the load on the interconnection network also increases.

We define traffic on a link $\mathcal{T}_\Gamma(e)$ as the number of messages that passes through the link e. Network traffic is the average traffic over all the links in an interconnect:

$$\mathcal{T}_\Gamma(e) = \sum_{\{u,v\}\in\mathcal{E}_\mathcal{G}} \frac{|\mathcal{S} = \{p : p \in \mathcal{P}(\Gamma(u), \Gamma(v)) \wedge e \in \mathcal{P}\}|}{|\mathcal{P}(\Gamma(u), \Gamma(v))|} \tag{2}$$

$$\mathcal{T}(\Gamma) = \frac{\sum_{e\in\mathcal{E}_\mathcal{H}} \mathcal{T}_\Gamma(e)}{|\mathcal{E}_\mathcal{H}|} \tag{3}$$

Finally, congestion on a link is defined as $\mathcal{X}(e) = \mathcal{T}_\Gamma(e)/c_\mathcal{H}(e)$ and network congestion is defined in terms of the maximum congestion on any of the network links, $\mathcal{X}(\Gamma) = \max_{e\in\mathcal{E}_\mathcal{H}} \mathcal{X}(e)$. Since a communication graph $\mathcal{G}$ does not contain any time related information, $\mathcal{X}(\Gamma)$ may not be a good approximation to the actual network congestion, in general. However, both $\mathcal{G}_{\text{col}}$ and $\mathcal{G}_{\text{row}}$ capture the communication happening in a small time window which begins and ends with a single collective call. Therefore, $\mathcal{X}(\Gamma)$ is a good approximation to the network congestions during the column and row communications of the eigensolver described in Sect. 2.

3.2 Practical Considerations

The performance results presented in Sect. 5 were obtained using the Hopper super-computer, which is a Cray XE6 machine at NERSC. Each compute node on Hopper contains 2 twelve-core AMD "MagnyCours" processors (24 cores per node) with 32 GBs of memory. Hopper uses the Cray "Gemini" interconnect for internode communication. The interconnection network has a 3D torus topology with dimensions 17×8×24. Two nodes share a single Gemini Network Interface Card (NIC), which has a total of 10 network connections, two each in `+x`, `-x`, `+z`, `-z`, and one `+y` and one `-y` links [3]. Consequently, the capacity of a link (c_H) in `+y` or `-y` direction is half the capacity of a link in other directions.

In order to compute the distance between two processing units, the physical coordinates of these units must be known. This machine-specific information can

be obtained through `xtprocadmin` and `xtdb2proc` utilities available in the Cray Linux Environment (CLE). Since Hopper's network has a 3D torus topology, there is a physical link between a node and its 6 neighbors located at `+x`, `-x`, `+y`, `-y`, `+z` and `-z` directions. So the number of links that a message needs to hop through is simply the Manhattan distance [1] (in 3D) between the physical coordinates of its start and destination nodes. On Hopper, a message is routed from the start node to the destination node through the unique shortest path using the links in the `x` dimension first, then the links in the `y` dimension, and finally those in the `z` dimension.

In Cray's MPICH2-based MPI library on Hopper, the collective operations of `MPI_Bcast` and `MPI_Reduce` are implemented using a binomial tree algorithm [5]. $\mathcal{E}_{\mathcal{G}_{\text{col}}}$ and $\mathcal{E}_{\mathcal{G}_{\text{row}}}$ are constructed by identifying the the binomial trees associated with the column and row communication groups of the triangular grid.

On Hopper, multiple processes would be mapped to the same physical node. As a result, Γ is a many-to-one function in this case. We assume that the intra-node communication bandwidth is infinite. Therefore, we do not include edges that correspond to intra-node communications in our model.

4 Heuristic for Task-to-Processor Mapping

Given a communication graph $\mathcal{G}$ and a physical interconnection network $\mathcal{H}$, the problem of finding a mapping $\Gamma : \mathcal{V}_\mathcal{G} \rightarrow \mathcal{V}_\mathcal{H}$ that minimizes the effective load on the network measured in terms of $\mathcal{D}(\Gamma)$, $\mathcal{T}(\Gamma)$ and $\mathcal{X}(\Gamma)$ is an NP-hard problem [1]. If we label the vertices in $\mathcal{V}_\mathcal{G}$ by 1, 2,..., n_p, constructing Γ is equivalent to assigning these numbers to processors placed on a triangular grid. For example, we may assign $1, 2, ..., n_\text{d}$ to the diagonal processors first, and continue the assignment for each of the subdiagonals until all processors on the grid are labeled. This gives what we call the diagonal-major (DM) ordering of the processors. Alternatively, we may go through the triangular processor grid column by column. This gives the column-major (CM) ordering. Row and column groups are created by grouping the numbered processing units based on their column and row positions. A clear drawback of this type grouping scheme is that the number of processing units in each row/column communication group is different, and the difference can be quite large. For example, while the largest C_col group contains n_d processing units, the smallest C_col group contains a single processing unit. As a result, there is a significant amount of imbalance in terms of communication volume among different communication groups.

To create a better mapping and grouping strategy, we extend the triangular grid to a square grid (note that $\hat{H}$ is symmetric), but require each processing unit to take either the (i, j)th grid point or the (j, i)th grid point, but not both. We modify the DM and CM orderings by limiting the number of processing units in each row or column of the grid to $(n_\text{d} + 1)/2$. If assigning a task number to the (i, j)th grid point violates this rule, we map the task to the (j, i)th grid point in the modified DM scheme which we refer to as balanced diagonal major

[1] See `http://en.wikipedia.org/wiki/Manhattan_distance` for a definition.

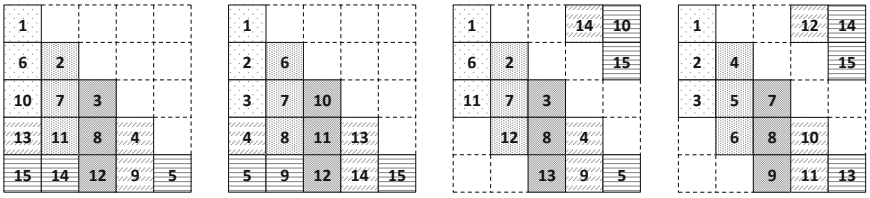

Fig. 4. Process orderings from left to right: DM, CM, BDM and BCM. Tasks mapped to the same column (row) of the grid belong to the same column (row) communication groups. Tasks with the same fill patterns belong to the same groups created for basis orthogonalization

(BDM) ordering. In the modified CM scheme, which we refer to as balanced column major (BCM) ordering, we start from the diagonal grid point in each column and assign a task to $(i - n_{\mathrm{d}}, j)$th grid point when $i > n_{\mathrm{d}}$. Figure 4 gives a schematic illustration of how DM, CM, BDM and BCM look for a 5×5 grid.

5 Performance Evaluation

In this section, we compare the mapping schemes described above on a few nuclear CI test problems involving ^{10}B. Different combinations of truncation ($N_{\max}$) and total magnetic angular momentum (M_j) parameters are used. Table 1 gives the size and sparsity characteristics for these problems and the number of cores used to solve these problems. The size of the problem (indicated by $nnz(\hat{H})$) increases roughly by a factor of 4 each time we change the $(N_{\max}, M_j)$ parameters. Since we are mainly interested in the weak scaling of MFDn, we increase the number of cores used by approximately a factor of 4, too.

Table 1. Matrix dimensions n and number of non-zero matrix elements nnz of the Hamiltonian $\hat{H}$ associated with nuclear structure calculations of ^{10}B using different parameter pairs ($N_{\max}$,M_j)

Test Name	($N_{\max}$, M_j)	$n(\hat{H})$	$nnz(\hat{H})$	n_{p}	$n(\hat{H})/n_{\mathrm{d}}$	$nnz(\hat{H})/n_{\mathrm{p}}$
test_{276}	(7,0)	4.66×10^7	2.81×10^{10}	276	2.0×10^6	1.1×10^8
test_{1128}	(8,1)	1.60×10^8	1.24×10^{11}	1128	3.4×10^6	1.1×10^8
test_{4560}	(9,2)	4.82×10^8	4.62×10^{11}	4560	5.1×10^6	1.0×10^8
test_{18336}	(10,3)	1.30×10^9	1.51×10^{12}	18336	6.8×10^6	0.9×10^8

5.1 Performance Results with Pure MPI Implementation

Our experiments were performed on Hopper. Figure 5 shows the observed communication overhead associated with DM, CM, BDM and BCM ordering schemes for all four test problems. In each case, the wallclock time spent for communication in DM (t_{DM}) is taken as the baseline (shown at 100%), and those associated with other ordering schemes are shown as percentages of t_{DM}. Each bar

in Fig. 5 shows three values: t_{col}, t_{row}, t_{orth}. They correspond to the wallclock times elapsed during communications within the column and row groups of the SpMV part and communication done for basis orthogonalization, respectively. We observe that the communication overhead of BDM is surprisingly higher than that associated with DM. We dropped the BDM scheme from larger test cases (test_{4560} and test_{18336}.) Both CM and BCM produce significant reductions in communication time compared to DM. The reduction ranges from about a factor of 2 for the smallest test-case to a factor of 5 for larger ones. Although keeping the same number of processing units in each communication group seems to be desirable, the improvement of BCM over CM is small in larger test cases.

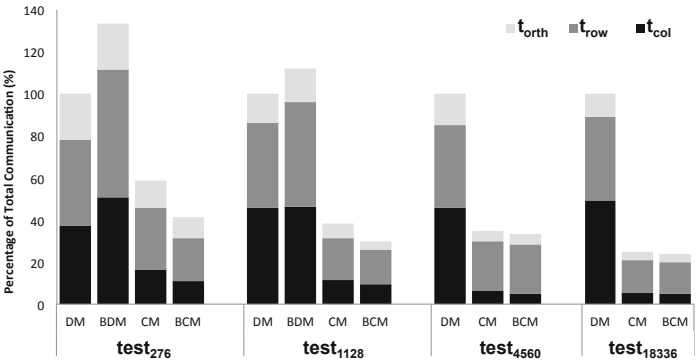

Fig. 5. Communication overhead associated with DM, CM, BDM and BCM ordering schemes with the pure MPI implementation

As discussed in Sect. 2, communication volume required in basis orthogonalizaton is relatively small compared to that required in SpMV. Therefore it is not surprising to see that t_{col} and t_{row} dominate the communication overhead in Fig. 5. In the largest test case involving 18,336 cores, running 99 Lanczos iterations takes 1,260 seconds when the DM ordering is used. On average 80% (or 1,010 seconds) of the total runtime is spent in communication. As can be seen, in the DM column of test_{18336} in Fig. 5, column group communications during SpMV is responsible for about 50% of the total communication time, and row group communications of SpMV is responsible for about 40%. The communication required in orthogonalization accounts for only 10%. Mapping tasks to processors according to the BCM ordering reduces the communication overhead by a factor of 5. While the overhead in all communication groups decreases sharply, the largest gain is seen in the column group, in which communication time drops from roughly 500 seconds for DM to only 50 seconds in BCM.

Since communication takes a significant portion of the total running time, the reductions in communication overhead achieved by BCM ordering translate to considerable speed-ups as summarized in Tab. 2. The last row in this table shows that the overall impact is as high as a factor of 2.57 speed-up in the total running time.

Table 2. Single-threaded performance improvement using different orderings

Ordering	Stats	test_{276}	test_{1128}	test_{4560}	test_{18336}
DM	t_{total} (sec)	211	410	567	1260
	$t_{\text{comm}}/t_{\text{total}}$	24%	47%	56%	80%
BDM	speed-up	0.93	0.95	–	–
	$t_{\text{comm}}/t_{\text{total}}$	30%	50%	–	–
CM	speed-up	1.12	1.41	1.57	2.52
	$t_{\text{comm}}/t_{\text{total}}$	16%	26%	31%	50%
BCM	speed-up	1.18	1.50	1.59	2.57
	$t_{\text{comm}}/t_{\text{total}}$	12%	21%	30%	49%

Table 3. Communication analysis for test_{276} and test_{1128}

		test_{276}		test_{1128}	
Ordering	Stats	C_{col}	C_{row}	C_{col}	C_{row}
DM	t_{comm} (sec)	19	21	89	78
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.9, 20, 56}	{0.8, 21, 62}	{5.5, 24, 148}	{5.6, 26, 136}
BDM	t_{comm} (sec)	26	31	90	97
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{1.1, 34, 86}	{1.1, 34, 86}	{7.5, 49, 163}	{7.4, 48, 167}
CM	t_{comm} (sec)	8.5	15	22	39
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.1, 3, 8}	{0.7, 20, 56}	{0.3, 5, 20}	{5.5, 23, 167}
BCM	t_{comm} (sec)	5.5	10.5	18	32
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.0, 0, 0}	{0.8, 14, 44}	{0.0, 0, 0}	{5.2, 20, 122}

Table 4. Communication analysis for test_{4560} and test_{18336}

		test_{4560}		test_{18336}	
Ordering	Stats	C_{col}	C_{row}	C_{col}	C_{row}
DM	t_{comm} (sec)	145	125	500	400
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{2.1, 97, 538}	{2.1, 98, 538}	{3.7, 121, 1489}	{3.6, 120, 1557}
CM	t_{comm} (sec)	20	75	55	155
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.1, 11, 25}	{2.1, 88, 290}	{0.1, 15, 48}	{3.6, 120, 632}
BCM	t_{comm} (sec)	15	75	50	150
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.0, 4, 5}	{2.4, 81, 330}	{0.0, 6, 20}	{3.6, 111, 626}

Tables 3 and 4 summarize the wallclock time used for communication within column and row groups for four different orderings schemes and two test problems each. All timing figures are accompanied by a triplet of numbers $\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$ that correspond to the dilation, average traffic and congestion metrics defined in Sect. 3.1. Note that message sizes vary significantly between test cases (see Tab. 1). Therefore we take the message size in test_{276} as our base unit and scale all metrics reported accordingly by dividing this message size to ensure a fair comparison across all test cases. A lower $\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$ value indicates lower load on

the network, hence lower communication overhead. For CM and BCM orderings, the network load due to communication within column groups is considerably less than that of row groups. This observation explains the lower communication overhead seen in column groups. In fact, this also indicates that our simple heuristic of mapping the processes in the same column communication group into "nearby" nodes actually works well.

In Fig. 5, it is not intuitive at first to see that BDM performs worse than DM. Even though the BDM ordering balances communication volume over all groups, Table 3 shows that the dilation associated with BDM is higher than that of the DM. Consequently, the increased network load in BDM leads to a poor overall performance. This observation suggests that topology-awareness is more important than simply keeping communication volume balanced among different groups when we construct a task-to-processor map.

As discussed above, computational load per processor is roughly the same across all test cases. However, as seen in Tab. 3 and 4, communication overhead increases sharply as the test problem becomes larger. This sharp increase is partially caused by a fast increase in communication volume (which is of magnitude $O(nn_d)$) which exceeds the linear increase in the network bandwidth with respect to n_p. However, network dilation and, perhaps more importantly, network congestion increases are also important factors to consider.

We can gauge the severity of the network congestion in the row communication group by comparing the $\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$ triplets associated with the DM ordering with that associated with the BCM orderings in test_{18336}. Even though dilation and average traffic seems to be roughly the same for both orderings, t_{comm} of DM ordering is significantly higher than that of BCM (400 vs. 150 seconds). So is its network congestion (1557 vs. 626).

5.2 Performance Results with the MPI/OpenMP Implementation

Table 5 compares the communication overhead of the pure MPI and hybrid MPI/OpenMP implementations of the Lanczos algorithm for the large test cases. Both implementations use the BCM ordering. Despite the reduction in communication volume, the communication time used in the column groups is much higher in the multi-threaded implementation for the test_{4560} problem. This is due to the increased dilation between communicating pairs in the column groups of the multi-threaded implementation. However, this difference is likely to vanish with

Table 5. Comparison of the pure MPI and hybrid MPI/OpenMP implementations for large testcases using approximately the same number of cores

		test_{4560}		test_{18336}	
Threading	Stats	C_{col}	C_{row}	C_{col}	C_{row}
single	t_{comm} (sec)	15	75	50	150
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.0, 4, 5}	{2.4, 81, 330}	{0.0, 6, 20}	{3.6, 111, 626}
multi	t_{comm} (sec)	40	60	58	110
	$\{\mathcal{D}, \mathcal{T}, \mathcal{X}\}$	{0.3, 24, 37}	{3.4, 44, 159}	{0.4, 29, 100}	{4.3, 89, 365}

increasing problem sizes, as indicated by the test_{18336} results. Multi-threaded implementation performs clearly better along the unoptimized C_{row} communicator, where the reduced number of messages and communication volume results in less traffic and congestion on the network.

6 Conclusions and Future Work

We developed topology-aware task-to-processor mappings to reduce the communication overhead in a parallel implementation of the Lanczos algorithm used to solve the nuclear many-body Schrödinger's equation. The effectiveness of a mapping can be assessed by examining the average network dilation ($\mathcal{D}$), average traffic ($\mathcal{T}$) and the maximum network congestion ($\mathcal{X}$) associated with the mapping. Each mapping corresponds to a particular ordering of the distributed tasks. We compared several mapping strategies and showed that the balanced column major (BCM) ordering of tasks gives the best performance through a number of computational experiments. This observation is consistent with our network load model which is defined in terms $(\mathcal{D}, \mathcal{T}, \mathcal{X})$. However, even in the case of BCM ordering, our optimization of the task-to-processor mapping is not performed globally among all processors. Therefore, we believe further improvement to the BCM ordering scheme is possible by applying topology mapping techniques described in [1,2] to all three communication groups created in our implementation of the Lanczos algorithm. We will focus on this approach in the future. Another factor that may affect the choice of an optimal mapping is the combination of thread-level parallelism with message passing based parallelism. This is an important issue for multi-core/many-core platforms.

Acknowledgments. This work was supported in part through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. DOE Office of Advanced Scientific Computing Research and Office of Nuclear Physics, by U.S. DOE Grants DE-FC02-09ER41582 (SciDAC/UNEDF) and DE-FG02-87ER40371, and by the US NSF grant 0904782. Computational resources were provided by NERSC, which is supported by the U.S. DOE Office of Science.

References

1. Hoefler, T., Snir, M.: Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In: Proceedings of the 2011 ACM International Conference on Supercomputing (ICS), Tucson, AZ (June 2011)
2. Bhatele, A., Gupta, G., Kale, L.V., Chung, I.-H.: Automated Mapping of Regular Communication Graphs on Mesh Interconnects. In: Proceedings of International Conference on High Performance Computing, HiPC (2010)
3. NERSC, Hopper, NERSC's Cray XE6 System (January 2012), Web. (February 15, 2012), `http://www.nersc.gov/users/computational-systems/hopper/`.
4. Demmel, J.: Applied Numerical Linear Algebra, 1st edn. SIAM (1997)

5. MPICH2, MPICH2: High-performance and Widely Portable MPI, http://www.mcs.anl.gov/research/projects/mpich2
6. Sternberg, P., Ng, E.G., Yang, C., Maris, P., Vary, J.P., Sosonkina, M., Le, H.V.: Accelerating Configuration Interaction Calculations for Nuclear Structure. In: The Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC 2008) (2008)
7. Maris, P., Sosonkina, M., Vary, J.P., Ng, E.G., Yang, C.: Scaling of ab-initio nuclear physics calculations on multicore computer architectures. Procedia CS 1, 97–106 (2010)