Integrated Link/CPU Voltage Scaling for Reducing Energy Consumption of Parallel Sparse Matrix Applications*

Seung Woo Son, Konrad Malkowski, Guilin Chen, Mahmut Kandemir, Padma Raghavan
The Pennsylvania State University
Department of Computer Science and Engineering
University Park, PA 16802 USA
{sson,malkowsk,guilchen,kandemir,raghavan}@cse.psu.edu

Abstract

Reducing power consumption is quickly becoming a first-class optimization metric for many high-performance parallel computing platforms. One of the techniques employed by many prior proposals along this direction is voltage scaling and past research used it on different components such as networks, CPUs, and memories. In contrast to most of the existent efforts on voltage scaling that target a single component (CPU, network or memory components), this paper proposes and experimentally evaluates a voltage/frequency scaling algorithm that considers CPU and communication links in a mesh network at the same time. More specifically, it scales voltages/frequencies of both CPUs in the network and the communication links among them in a coordinated fashion (instead of one after another) such that energy savings are maximized without impacting execution time. Our experiments with several tree-based sparse matrix computations reveal that the proposed integrated voltage scaling approach is very effective in practice and brings 13% and 17% energy savings over the pure CPU and pure communication link voltage scaling schemes, respectively. The results also show that our savings are consistent with the different network sizes and different sets of voltage/frequency levels.

1. Introduction

Power consumption is becoming a critical issue for high-end computing platforms due to several factors including costs, space, reliability, and maintenance. Consequently, recent research efforts from different groups in both academia and industry have focused on techniques that help us accurately model and reduce power consumption of different hardware components in a large computing infrastructure. These studies, details of which are discussed in Section 2, include CPU power optimizations, memory banking and low-power operating mode management, network power minimization, and energy-oriented disk I/O optimizations.

*This work is supported in part by NSF grants CCF 0444158, CNS 0406340, CCF 0444345, and CCF 0102537.

Voltage and frequency scaling has been identified by past research as one of the most effective ways of reducing CPU power [10, 28]. More recently, there have been proposals [25, 29] that apply voltage/frequency scaling to network links to save communication power. However, to our knowledge, none of the prior efforts in the domain of high-performance computing considered using voltage scaling on both CPUs and communication links of a given parallel architecture in a coordinated fashion to save power. The work described in this paper is a step in this direction. More specifically, focusing on sparse matrix computations that can be represented as trees, this paper studies the potential benefits that can be accrued when using CPU and communication link voltage/frequency scaling in a coordinated fashion. To achieve this, we propose and experimentally evaluate a voltage/frequency scaling algorithm.

An important characteristic of the proposed algorithm is that it tunes the CPU and link voltages carefully so that we can obtain the potential power savings without increasing the original execution time, i.e., the time taken when no voltage scaling is employed. The important point is that we scale the voltages of CPUs and links considering the impact of doing so on each other; this is radically different from an alternate approach that applies CPU voltage scaling after communication link voltage scaling or vice versa. To test the effectiveness of our approach, we applied it to a set of tree-based sparse matrix computations running on a two-dimensional mesh network and compared it two alternate schemes, one that applies voltage scaling only to CPUs and the other one that applies voltage scaling to only communication links. Our experiments reveal that the proposed integrated voltage/frequency scaling approach is very effective in practice and brings 13% and 17% energy savings over the pure CPU and pure communication link voltage scaling schemes. The results also show that our savings are consistent with the different network sizes and different sets of voltage/frequency levels.

The remainder of this paper is structured as follows. In the following section, we describe the related work on voltage scaling in the context of the interconnection network and processors. Section 3 explains the tree based computation model for parallel sparse matrix solvers. Our inte-

1-4244-0054-6/06/$20.00

grated link/CPU voltage scaling algorithm is presented in Section 5. Section 6 presents an experimental evaluation of the proposed algorithm. We conclude the paper in Section 7 with a summary of our major contributions.

2. Related Work

Several studies in the past have proposed dynamic voltage scaling (DVS) techniques for reducing energy consumption of communication links in the NoC (Network-on-Chip) based systems and high-end multiprocessor systems [25, 26, 29]. The main idea behind these approaches is to scale down the voltage/frequency of communication links when there is enough communication slack (i.e., the amount of latency by which communication can be delayed without affecting overall execution time) observed or predicted. In order for these DVS techniques to be feasible, Kim et al [19] proposed serial links that can operate under various link voltage/frequency levels. Employing links with variable voltage/frequency, Shang et al [25] presented and evaluated a history-based DVS scheme for the communication links. Worm et al [29] proposed an adaptive low-power transmission technique for on-chip networks, whereas Shin et al [26] discussed a task mapping technique based on genetic algorithms to utilize voltage scalable links for saving energy in NoC based systems. Besides DVS techniques for communication links, several techniques that shut down unused or underutilized links have proposed. Kim et al [18] proposed a dynamic link shutdown (DLS) technique for chip-to-chip networks. Soteriou et al [27] explored the design space for communication links with turn on/off capability.

In addition to these efforts that target at reducing power consumption in communication links, there are also studies that target at reducing power/energy consumption of large server and cluster systems. These efforts can be broadly classified into three categories. The first category of the efforts considered CPU-centric techniques that turn off unused CPUs [8] or scale down CPUs that execute non-critical execution [9, 10]. Voltage scaling on processors [28] has been extensively studied and several commercial processors (e.g., Transmeta's Crusoe [1] and AMD's Athlon 64 [2]) already provide mechanisms to control the frequency and voltage of processors. The second category of studies [3, 18, 5] proposed several techniques that focus on individual components of the server based computing systems such as CPUs and main memory. Lastly, many studies focused on reducing energy consumption on the disk subsystem, which is a huge energy consumer for large data centers, by completely spinning down disks [7] or dynamically adjusting the rotational speed of disks [14].

Lastly, Chen et al [4] recently proposed a CPU voltage scaling scheme for tree based parallel sparse matrix applications. Our approach is different from their approach because we scale down the voltage/frequency of both CPUs and links at the same time. In the domain of real-time distributed embedded systems, Luo et al [20] proposed a technique that simultaneously scales voltages of processors and communication links. Our approach is also different from Luo et al's work in that we focus on exclusively parallel sparse matrix applications and consider the underlying network topology in selecting proper link voltages (and corresponding frequencies). Consequently, our integrated voltage scaling algorithm is entirely different from theirs.

3. Tree-Based Computation Model

In this paper, we concentrate on parallel sparse linear systems to study the impact of CPU and link voltage scaling without impacting performance. Such computations typically dominate the execution time of many large-scale parallel applications on multiprocessors and clusters of workstations. There are many classes of parallel sparse linear solvers and important classes include parallel direct solvers based on sparse factorization [6, 12, 17, 22], iterative solvers [16, 24], and direct-iterative hybrids through preconditioning [13, 21, 23]. While there is no single method that is always better than others across the different application domains and the underlying parallel execution platforms, they share the same notion that a given sparse matrix can be represented as a graph, which can be partitioned across processors for parallel execution. This partitioning [17] is usually performed using a recursive scheme for computing vertex or edge separators, and the associated partitioning tree (and related trees) can serve as a useful model for the underlying parallelism and data dependencies.

We focus on tree-based parallel sparse computations that are representative of parallel sparse solvers when matrix is symmetric positive definite. Such solvers consist of a initial symbolic phase followed by a numeric phase. In the symbolic phase, the matrix for parallel computation is partitioned to determine the actual structure of the Cholesky factor L [15]. The numeric phase, which represents the dominant cost in total solver time, involves computation of the sparse factor and solving the problem using the determined sparse factor. The columns of L can be clustered into *supernodes*, each of which contains a set of consecutive columns with the same zero/nonzero structure. The overall numeric phase can be performed in parallel on *tree of supernodes.* The tree structure represents the data dependencies, and each tree node denotes a supernode of L and its corresponding set of dense-matrix operations. The allocation of processors to subtrees is based on the weights on the tree to represent the computation costs. While the allocation procedure can be done in several phases to balance computational load on each processor, inherent irregularities in the sparse matrix often result in workload imbalance across processors during parallel sparse matrix computation.

In this paper, we use a *weighted tree* as the model of sparse computation. An example weighted tree is given in Figure 1. We use N_i to represent the nodes in the tree. Each leaf node, which represents a local computation phase, is assigned to one processor, p_i, as shown in Figure 1. The nodes, which represent the distributed phase, are assigned to the processors that also operate to the leaf nodes of their subtree nodes. For example, N_1 is assigned to 4 processors, from p_0 to p_3, since N_1 makes use of all the processors in that subtree. The pair of numbers inside a given node in this figure represents the computation and commu-

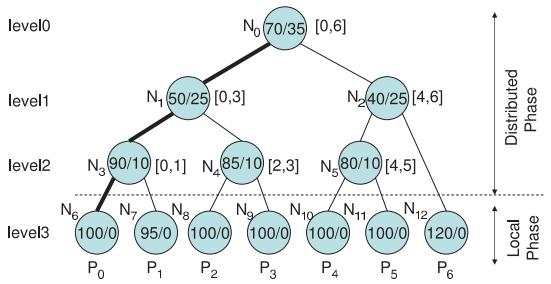


Figure 1. An example tree representing parallel sparse computations.

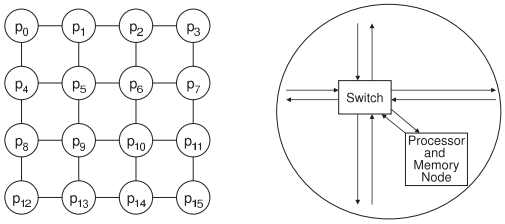


Figure 2. (a) Two-dimensional mesh topology (4 by 4). (b) A node with a bidirectional network.

nication weights associated with that node, e.g., N_1 is assigned 50 unit of computation load and 25 unit of communication load. Similarly, N_3 is assigned 90 and 10 units of computation and communication loads, respectively. In the tree-based computation model for sparse applications, the weight on each node is evenly distributed across all the processors assigned to that tree node. Based on this weighted tree notation, we can determine the *critical path*, which is represented as thick solid line in Figure 1. In other words, N_0, N_1, N_3 and N_6 constitute the critical path in this example tree, which determines the minimum execution time of the parallel application. The goal of this study is to reduce energy consumption of such tree-based matrix computations by taking advantage of the computation and communication exhibited by the tree nodes not in the critical path.

4. System Model

We focus on an $M \times N$ two-dimensional mesh based interconnection network as shown in Figure 2(a), though the analysis described in this work is applicable to other network topologies as well with proper modifications. Each pair of adjacent nodes in this 2D mesh topology is connected to each other using two uni-directional links. A node in this architecture typically consists of one or more processors, some amount of local memory, and a switch that routes messages through the nodes (Figure 2(b)). We use p_i to denote the id of the i^{th} node in this mesh network, which can be written as:

$$p_i = row(i) \times M + col(i), \quad (1)$$

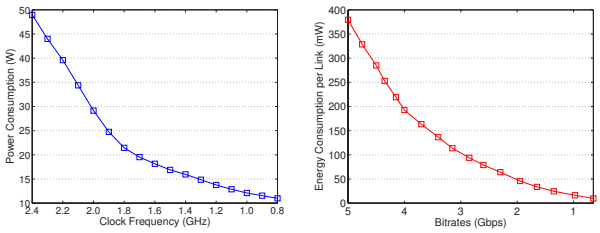


Figure 3. Left: CPU power model. Right: Link power model.

where M is the row size of the mesh and $row(i)$ and $col(i)$ are the row and column positions, respectively, of the i^{th} node. For example, p_5 in Figure 2(a) can be represented using $(1, 2)$ since the size of this example mesh topology is 4×4.

Given the system architecture explained above, an application program considered in this study is parallelized using the message-passing interface, MPI [11]. In this model, the nodes communicate with each other using explicit send/receive commands. We propose integrated voltage/frequency scaling on CPUs and communication links. We apply our technique to these two components due to following reasons. First, both modern CPUs and communication links support voltage/frequency scaling circuits and we can make use of these capabilities to reduce power. Another reason is that each of these components are known to be a major contributor to total energy consumption in large parallel machines [4, 20].

To illustrate the idea behind our approach that scales both CPU and communication link voltages in a coordinated fashion, let us first take a look at their energy/latency behavior. The dynamic power consumption, P, can be represented as:

$$P = 1/2 \cdot f \cdot N \cdot C \cdot V_{dd}^2, \quad (2)$$

where f is clock frequency, N is switching activity, C is effective capacitance, and V_{dd} is supply voltage. Therefore, the power consumption, P, is quadratically proportional to the supply voltage, V_{dd}, and frequency, f[1]. As shown by Equation 2, we can obtain quadratic drop in power consumption with a linear reduction in the clock frequency (f) and the supply voltage (V_{dd}).

The energy versus clock frequency[2] curves for CPU and communication links are drawn in Figure 3. The CPU curve is adapted from AMD's Athlon-64 processor datasheet [2], whose clock frequency can range from 800MHz to 2400MHz and the corresponding supply voltage can range from $1.1V$ to $1.5V$[3]. The communication link curve on

[1] Since the clock frequency, f, can be represented in terms of V_{dd} and threshold voltage, V_t, as the frequency is reduced, the supply voltage can be reduced proportionally.

[2] In case of a serial link, the frequency dictates the bit-rates.

[3] Since the AMD datasheet states only TDP (Thermal Design Power), which is 89 W, we estimate the peak power consumption of the CPU for our study to be approximately 50 W based on our experience.

the other hand is generated using data collected from [19], which has a supply voltage range of $0.9V$ to $2.5V$. The corresponding bit-rate range is from 650Mb/s to 5Gb/s. We can see from this figure that frequency-power curves are *convex*, which means that, as voltage and frequency are scaled down, the additional energy savings gained drop quadratically. Therefore, it should be beneficial from the energy perspective to scale voltages/frequencies of both CPU and link in a balanced (coordinated) manner, instead of scaling one of them aggressively.

5. Integrated CPU and Link Voltage Scaling Algorithm

We now explain our algorithm for simultaneous voltage/frequency scaling for CPUs and communication links in tree-based parallel sparse computations using the example given in Figure 1, which is actually extracted from one of programs we tested in our experiments. The goal of our algorithm is to find an appropriate voltage levels and the corresponding frequency levels for CPUs and links that maximizes energy savings without impacting the overall execution time. Since our computation model is based on a tree, we can apply one of algorithms proposed by Chen et al [4] where only CPU voltage is scaled down to save energy. Our algorithm starts with same approach as given in that, but it needs to be applied carefully due to the conflicts between the different voltage levels chosen for the communication links. To better explain this, let us consider the communication patterns exhibited by the example computation tree shown in Figure 1. Note that, since the leaf nodes in our tree-based sparse computation model perform only computation, the communication starts from the nodes just above the leaf nodes, i.e., level 2 in the tree. One notable characteristic of communicating nodes is that they can be grouped using a neighborhood concept. We use a CG (*Communication Group*) to represent the nodes that participate in communication at any point during execution, and a CG can be represented as follows:

$$[p_{low}, p_{high}], \tag{3}$$

where p_{low} is the processor whose id is the smallest among the processors in a given CG, and p_{high} is the processor whose id is the largest among the processors. Additionally, we use *size(CG)* to capture the number of processors in the CG. For example, in level 2 of the tree in Figure 1, there are three CGs (CG_3, CG_4, and CG_5), which can be represented as [0,1], [2,3], and [4,5], respectively. The size of all three CGs are 2 in this case. As we can see from this figure, the size of CGs becomes larger as we move to the lower levels, i.e., towards the root of tree. To see how these CGs are mapped onto the underlying mesh topology, let us consider the mapping between one of detected CGs, CG_4 in the level 2 of Figure 1, and the mesh topology, which is illustrated in Figure 4. Since we use an X-Y routing algorithm in communicating among nodes, each CG can be mapped to a set of rectangularly-connected nodes in the mesh topology. Therefore, we need to adjust p_{low} and p_{high} values of each

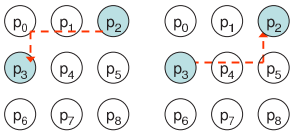


Figure 4. Mapping CG_4 to mesh topology.

Table 1. Example voltage/power levels.

Voltage	Power
1	1
0.8	0.512
0.6	0.216
0.4	0.064

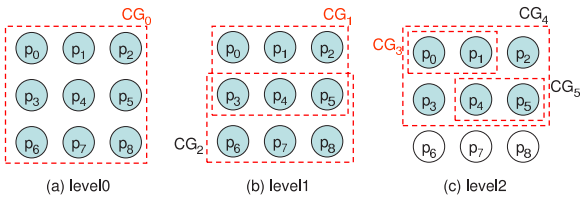


Figure 5. CGs mapped to the mesh topology.

CG using the actual processor-to-mesh topology mapping, and this can be represented as follows:

$$p'_{low} = (min(row(\forall p_i)), min(col(\forall p_i))),$$
$$p'_{high} = (max(row(\forall p_i)), max(col(\forall p_i))), \tag{4}$$

where min() and max() functions give the minimum and maximum values of the column and row indices, respectively, of all the processors in each CG. Using this equation, we can obtain CG_4 mapped to the underlying mesh topology. Initially, CG_4 can be represented as [2,3]. As illustrated in Figure 4, CG_4 needs additional 4 processors (p_0, p_1, p_4, and p_5) to communicate each other using the X-Y routing algorithm. Since the row and column index for p_2 and p_3 are (0,2) and (1,0) respectively, we can redefine CG_4 as (0,0) and (1,2) using Equation (4) given above. All other detected CGs are marked using dashed rectangles in Figure 5 for each level of our example tree.

In the first step of our algorithm, we build a CG-annotated tree of the given parallel sparse matrix computation. Recall that, when we are given a tree representation of parallel sparse matrix computation such as the one in Figure 1, we already know the particular tree nodes that reside on the critical path of the tree. After obtaining all CGs, we then move to determine *conflict group*, denoted as D in this paper, to capture whether there is any conflict among the CG groups that sit in the same level. We say that there is conflict between nodes CG_x and CG_i if the following condition holds true:

$$(D = CG_i \cap CG_x) \neq \emptyset \wedge \text{level}(CG_i) = \text{level}\ (CG_x), \tag{5}$$

where CG_x is a CG in the critical path. In Figure 5(b), CG_1 and CG_2 are in conflict because three processors, namely p_3, p_4, p_5, are shared by both CG_1 and CG_2. Note that the root node is always in the critical path so that all the nodes should operate under the maximum available frequency, i.e., maximum voltage level supported by the architecture (see Figure 5(a)) when they work on the root node.

```
VS_main (node) {
   build_CG (node);
   voltage_scale (node);
}

build_CG (node) {
   if (node is leaf) {
      node.plow = node.phigh = node.processor;
      CG = [node.plow, node.phigh];
   }
   else {
      node.plow = ( min (row(∀p_i ∈ CG)), min(col(∀ p_i ∈ CG)) );
      node.phigh = ( max (row(∀p_i ∈ CG)), max(col(∀ p_i ∈ CG)) );
      CG = [node.plow, node.phigh];
      build_CG (node.left);
      build_CG (node.right);
   }
}

voltage_scale (node) {
   if (node is leaf) {
      // assign the lowest link power level
      node.link_level = MIN_LINK_LEVEL;    return;
   }
   else { // node is not leaf, i.e., node has children
      // determine critical node
      if (node.left.total_t < node.right.total_t ) {
         scale_node = node.left;    critical_node = node.right;
      }
      else {
         scale_node = node.right;    critical_node = node.left;
      }
      D = CG_i ∩ CG_x; // CG_x is the CG in critical path
      // select the slowest level for both CPU and link simultaneously.
      while ( (curr_cpu_level < MIN_CPU_LEVEL) &&
            (curr_link_level < MIN_LINK_LEVEL) &&
            (node.time < node.total_t)) {
         curr_cpu_level- -;
         ∀p_i ∉ D curr_link_level- -;
      }
      // communication is dominant → reduce CPU voltage further
      if (scale_node.orig_tot_comm_t > scale_node.orig_tot_comp_t) {
         while ((curr_cpu_level < MIN_CPU_LEVEL) &&
            (node.time < node.total_t)) {
            curr_cpu_level- -;
         }
      else { // computation is dominant → reduce link voltage further
         while ((curr_link_level < MIN_LINK_LEVEL) &&
            (node.time < node.total_t)) {
            curr_link_level- -;
         }
      }
      node.cpu_level = curr_cpu_level;
      node.link_level = curr_link_level;
      // recalculate total time based on newly determined voltage
      // levels and update node with the scaled voltage/frequency
      voltage_scale (node.left);
      voltage_scale (node.right);
   }
}
```

Figure 6. Integrated CPU/link voltage/frequency scaling algorithm.

For level 1, we are able to reduce the voltage levels of processors that belong to CG_2, which is not in the critical path, and do not belong to the set of conflicting processors in CG_2. In our example, we can reduce the voltage levels of processors p_6, p_7, and p_8. It should be mentioned that the slack in a given node must be large enough to scale both CPU and link voltages. Once we assign determined voltage levels, we then recalculate the slack at each node in the tree. Our algorithm continues this way until all the nodes of the tree are processed. Note that, at the leaf nodes, we scale down all link voltage to the lowest levels because no communication is involved at leaf nodes.

The algorithm given in Figure 6 follows the approach explained above. Basically, our algorithm scales down a subtree, which is not in the critical path, as a whole. The algorithm starts by generating the CG-annotated tree by invoking the build_CG function, followed by calling the voltage_scale function. This function is invoked recursively,starting from the root node. If the node currently being processed is a leaf, our algorithm assigns the lowest voltage levels to all the communication links in the mesh. If the input node has a child node and one of its subtrees has slack, we scale down both CPU and link voltages at the same time until the point where the scaled execution time becomes very close to the original execution time. After scaling voltages simultaneously, if the remaining slack is large enough to scale down either CPU or link voltages, we further apply voltage scaling on that node. The decision on whether to scale down CPU voltage or link voltage is made based on their contribution to the total execution time of that node. More specifically, if the total computation time is longer than the total communication time, we scale down the link voltage because scaling down the component whose contribution is larger tends to consume the observed slack more quickly. So, it is better to scale the link voltage from the energy perspective while utilizing the slack efficiently. On the other hand, in the case where communication is dominant time consumer for the node being processed, we scale down the CPU voltage. Our algorithm continues in this fashion until all the nodes of the tree are processed.

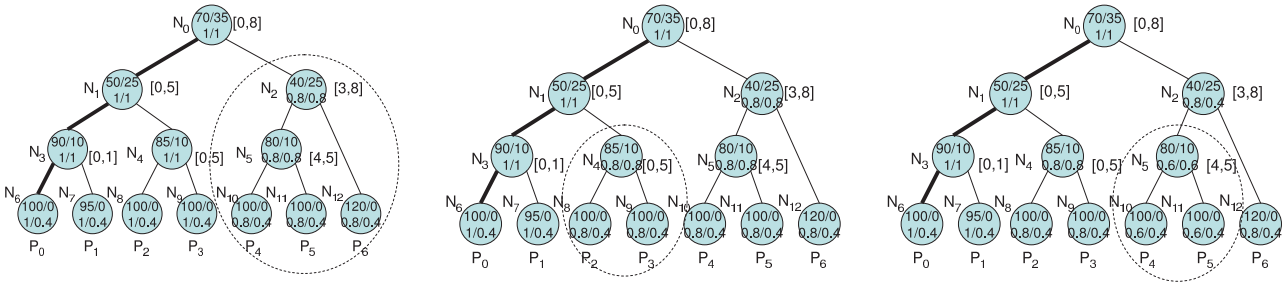
Figure 7 shows how our approach works in practice. For illustrative purposes, we use the normalized voltage and power numbers for both the CPU and link, which are given in Table 1. The initial voltage/power numbers are 1/1, as shown inside each node in Figure 7. In the first phase, we scale down the right subtree because the left subtree is in the critical path. Therefore, we scale the both link and CPU voltage of all nodes in the right subtree to one level lower, 0.8 in this example. Note that, the link voltage of all leaf nodes are set to the lowest levels because this does not increase execution time. In the subsequent phase, our approach scales down the voltage/frequency of the subtree whose root is N_4 (Figure 7(b)). Lastly, the subtree rooted at N_5 can be scaled down further by using the slack present in that subtree (Figure 7(c)).

6. Empirical Evaluation

6.1. Experimental Setup

To evaluate our approach, we implemented a simulation platform shown in Figure 8. We first obtain trace data, which indicate the computation weight involved at each level of the tree, the CGs detected, and the communication patterns, from parallel sparse matrix solver. This trace data is then fed to the energy simulator along with the CPU and link power models. Based on the given voltage scaling mode, which will be explained shortly, the energy simulator generates energy and performance statistics.

To obtain the energy consumption of CPUs and network links, we use an energy model similar to that described in the literature [2, 19]. The default simulation parameters used in our experiments are given in Table 2. By default,



(a) 1st phase. (b) 2nd phase. (c) 3rd phase.

Figure 7. Example application of our approach. The dashed circle is the subtree nodes being scaled in the corresponding phase.

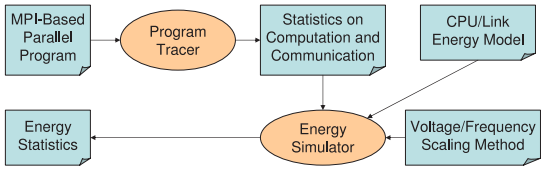


Figure 8. Our simulation platform.

we have 5 voltage levels for both CPU and link. We further increase the number of voltage levels later in this paper to see the trends in energy savings with varying number of voltage levels.

The parallel sparse matrix solvers experimented in this study are given in the first seven rows of Table 3. In addition to these practical solvers, we also experimented with 5 model sparse matrices, which are given in the last 5 rows of Table 3, to study the sensitivity of our approach to the increased problem size. The number of computing nodes (i.e., processors) and the size of mesh network used in each solver are given in the second and third columns of Table 3, respectively. The fourth and fifth columns of the table show the number of messages communicated among processors during computation and the total data volume of the communicated messages, respectively. Note that, the numbers given in these two columns are the average weight per processor. So, the total weight is dependent on the number of processors involved in each tree node. The last column is the contribution of the communication time to the total execution time. We can see from this table that the communication time (correspondingly communication volume and the number of messages) increases when more processors are involved in parallel execution. Since we use square mesh networks, our energy simulator takes into account the energy consumption of the CPUs and links that are actually used.

To evaluate the effectiveness of our approach, we conducted experiments with the following three schemes:

- CPU-VS: This scheme scales down only CPU voltages, using the algorithm proposed by Chen et al [4]. It simply takes advantage of available computation slacks.

Table 2. Default simulation parameters.

Parameter	Value
CPU clock frequency range	800MHz $\sim$ 2400MHz
Number of voltage/frequency levels	5
Supply voltage range	1.1 $\sim$ 1.5V
Default clock frequency	2400 MHz
Frequency transition latency	1 ms
Link frequency range	130MHz $\sim$ 1GHz
Number of voltage/frequency levels	5
Number of multiplexing stage	5
Bitrates per link	650Mb/s $\sim$ 5Gb/s
Link supply voltage range	0.9 $\sim$ 2.5V
Active link energy consumption	10.2 pJ/bit
Idle link energy consumption	8.5 pJ/cycle
Link frequency transition latency	10μs (100 link cycles)

- LINK-VS: This scheme uses the same algorithm proposed by Chen et al [4] except that it is applied to scale down only link voltages based on the communication slacks available. The selection of link voltage level is made based on the algorithm explained in Section 5.

- CPU-LINK-VS: This scheme, which is the main contribution of this work, scales both CPU and link voltages using the algorithm given in Figure 6. If there is enough slack, this scheme tries to scale down both CPU and link simultaneously. When a voltage level chosen for one CG is not the same as those of the other CGs that share processors for communication, CPU-LINK-VS chooses the largest voltage level among the voltage levels of all the CGs, in an attempt to minimize potential performance overheads.

6.2. Results

Figure 9 gives the normalized energy savings with the three different schemes described in Section 6.1. All bars of a given solver are normalized with respect to the execution when *no* voltage/frequency scaling is applied. We can see from this figure that the energy savings obtained from both CPU-VS and LINK-VS are significant. Specifically, the average energy savings by CPU-VS and LINK-VS are 27% and 23%, respectively. This shows that scaling down either CPU or link voltage can be very effective in reducing total energy consumption. On the other hand, the CPU-LINK-VS scheme, which scales down CPU and link in a coordi-

Table 3. Set of parallel sparse solvers.

Solver Name	Number of Processors	Mesh Dimensions	Number of Messages	Communication Volume (MB)	Percentage of Communication Time
bmw7st1	64	8×8	24,337	406.41	50.7%
bcsstk31	28	6×6	4,645	18.36	31.3%
bcsstk35	17	5×5	6,999	43.13	22.3%
crystk02	11	4×4	2,227	7.05	9.8%
finan512	28	6×6	7,364	39.59	44.7%
nasasrb	22	5×5	2,997	10.13	6.1%
tube1	7	3×3	2,557	12.16	8.7%
205x205	3	2×2	291	0.29	0.2%
256x256	7	3×3	648	0.82	3.8%
320x320	15	4×4	1,294	2.1	22.6%
400x400	31	6×6	2,318	4.5	38.1%
500x500	63	8×8	3,172	7.1	39.1%

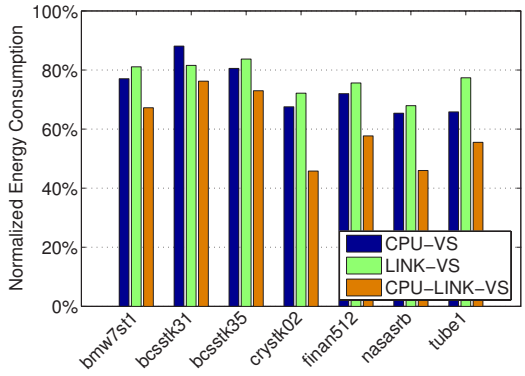


Figure 9. Normalized energy consumptions with the different schemes.

nated fashion, achieves 40% energy saving on average. This result clearly shows that it is better to scale voltages of both CPUs and links in an integrated manner rather than scaling only one of them aggressively, due to the diminishing energy saving rates, already demonstrated earlier by Figure 3. Note that, since all three schemes try to scale down the voltages/frequencies of the tree nodes that are not in the critical path, no schemes incurs any observable performance degradation.

In our next set of experiments, we perform a sensitivity analysis to see how the energy savings achieved by our approach are affected with the increase in the number of voltage/frequency levels supported by the underlying architectures, and the number of processors. To study the effectiveness of our approach with finer voltage levels in CPU and links, we experiment with 5 (our default value), 9, 17, and 33 voltage levels. The intermediate voltage levels are obtained by curve fitting based on the initial voltage/frequency points. All other simulation parameters are fixed as in Table 2. The normalized energy savings for all seven solvers used in our experiment under the different number of voltage levels are given in Figure 10. As one can observe from these graphs, the energy savings obtained saturate as we increase the number of voltage/frequency levels. This is an anticipated result since finer granular voltage levels give more opportunity to scale down voltage levels, even when we have small slacks. However, we also see that energy savings start to saturate when the number of voltage levels reaches 17 or so. This shows that our scheme makes use of slacks in the tree successfully with reasonable number of voltage/frequency levels.

In the next set of experiments, we vary the number of processors and, correspondingly, the size of our two-dimensional mesh topology. We used the model solver in this experiment with five different processor sizes: 3, 7, 15, 31, and 63. The model solvers are generated by increasing the problem size with the increased number of processors. Recall that we do not consider the energy consumption of the unused CPU nodes and the communication links connected to them. The normalized energy consumption with various processor sizes are given in Figure 11. We can see from this figure that, as we increase the number of processors, the energy savings achieved by all three schemes decrease. The reason why the energy savings achieved by CPU-VS decrease is that, as the number of processors increases, overall execution time is dominated by communication, thereby decreasing the opportunities for scaling down the CPU voltages. Similarly, the energy savings achieved by LINK-VS also decrease due to the increased network contention brought by the larger number of processors, and network topology prevents the possibility to scale down the link voltages. Lastly, the energy savings obtained through the CPU-LINK-VS also decreases but this scheme gives the best energy savings for the all fives cases tested.

7. Conclusions

The main contribution of this paper is an algorithm that scales voltages/frequencies of CPUs and communication links in a mesh-based parallel system in a coordinated (integrated) fashion such that energy savings are maximized and performance is not affected. To test our algorithm, we implemented it and applied it to a set of tree-based sparse computations. The experimental results collected are very promising and show that integrated CPU/communication link voltage scaling can generate much better results than the CPU voltage scaling alone and the link voltage scaling alone. Our results also show that the energy savings are consistent with the different problem sizes and different sets of voltage/frequency levels.

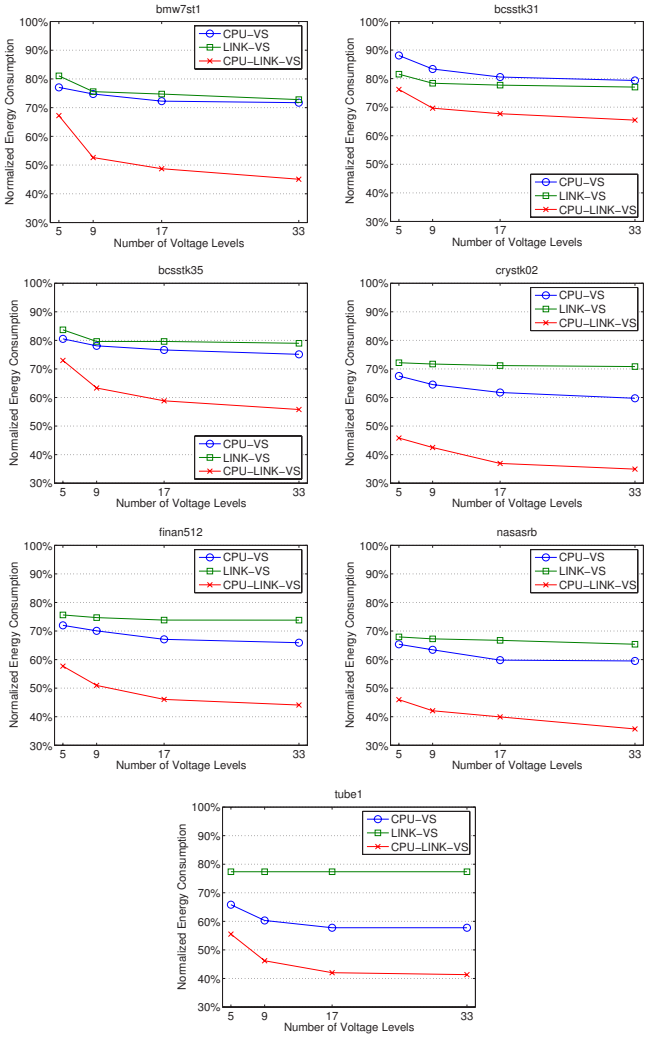


Figure 10. Normalized energy consumptions with the different schemes as the number of voltage/frequency levels vary.

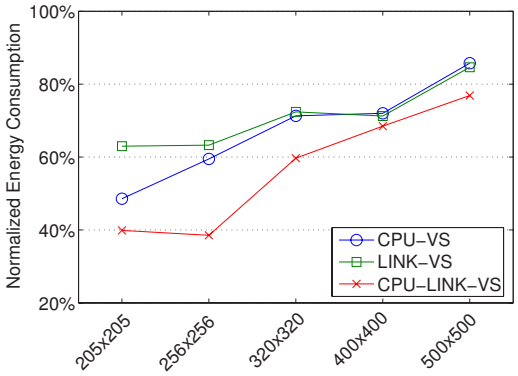


Figure 11. Normalized energy consumption as the number of processors increase.

References

[1] Crusoe Longrun Power Management White Paper. http://www.transmeta.com/crusoe/longrun.html.

[2] Advanced Micro Devices, Inc. AMD Athlon 64 Processor Power and Thermal Data Sheet, 2004.

[3] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th Symposium on Operating Systems Principles*, pages 103–116, October 2001.

[4] G. Chen, K. Malkowski, M. T. Kandemir, and P. Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *Proceedings of International Parallel and Distributed Processing Symposium*, April 2005.

[5] X. Chen and L. Peh. Leakage power modeling and optimization in interconnection networks. In *Proceedings of the International Symposium on Low Power and Electronics Design*, pages 90–95, August 2003.

[6] J. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. Technical Report CSL–94–14, Xerox Palo Alto Research Center, 1995.

[7] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *Proceedings of the USENIX Winter Conference*, pages 292–306, 1994.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient Server Clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.

[9] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[10] V. W. Freeh and D. K. Lowenthal. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 164–173, 2005.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[12] A. Gupta, F. Gustavson, M. Joshi, G. Karypis, and V. Kumar. PSPASES: An Efficient and Scalable Parallel Sparse Direct Solver, 1999. http://www-users.cs.umn.edu/~mjoshi/pspases.

[13] A. Gupta, V. Kumar, and A. Sameh. Performance and Scalability of Preconditioned Conjugate Gradient Methods on the CM-5. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 664–674, 1993.

[14] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the International Symposium on Computer Architecture*, pages 169–179, June 2003.

[15] M. T. Heath, E. Ng, and B. W. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33:420–460, 1991.

[16] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[17] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[18] E. J. Kim, K. H. Yum, G. Link, C. R. Das, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Energy Optimization Techniques in Cluster Interconnects. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 459–464. ACM, August 2003.

[19] J. Kim and M. A. Horowitz. Adaptive Supply Serial Links with sub-1V Operation and Per-pin Clock Recovery. In *Proceedings of International Solid-State Circuits Conference*, February 2002.

[20] J. Luo, L.-S. Peh, and N. Jha. Simultaneous Dynamic Voltage Scaling of Processors and Communication Links in Real-time Distributed Embedded Systems. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 1150–1151, 2003.

[21] E. Ng and P. Raghavan. Towards A Scalable Hybrid Sparse Solver. *Concurrency: Practice and Experience*, 12:1–16, 2000.

[22] P. Raghavan. DSCPACK: Domain-Separator Codes for the parallel solution of sparse linear systems, 2002.

[23] P. Raghavan, K. Teranishi, and E. Ng. A latency tolerant hybrid sparse solver using incomplete cholesky factorization. *Numerical Linear Algebra*, 10:541–560, 2003.

[24] Y. Saad. *Iterative Methods for Sparse Linears Systems*. PWS Publishing Co., Boston, MA, 1996.

[25] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pages 91–102, 2003.

[26] D. Shin and J. Kim. Power-Aware Communication Optimization for Networks-On-Chips with Voltage Scalable Links. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 170–175, 2004.

[27] V. Soteriou and L.-S. Peh. Design-Space Exploration of Power-Aware On/Off Interconnection Networks. In *Proceedings of the IEEE International Conference on Computer Design*, pages 510–517, 2004.

[28] M. Weiser, A. Demers, B. Welch, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of Symposium on Operating System Design and Implementation*, pages 13–23, Nov. 1994.

[29] F. Worm, P. Ienne, P. Thiran, and G. D. Micheli. An Adaptive Low-Power Transmission Scheme for On-Chip Networks. In *Proceedings of the 15th international symposium on System Synthesis*, pages 92–100, 2002.