A Novel Application of Parallel Betweenness Centrality to Power Grid Contingency Analysis

Shuangshuang Jin, Zhenyu Huang, Yousu Chen, Daniel Chavarría-Miranda, John Feo, Pak Chung Wong
Pacific Northwest National Laboratory,
Richland WA 99354, USA
e-mail: {shuangshuang.jin, zhenyu.huang, yousu.chen, daniel.chavarria, john.feo, pak.wong}@pnl.gov

***Abstract*—In Energy Management Systems, contingency analysis is commonly performed for identifying and mitigating potentially harmful power grid component failures. The exponentially increasing combinatorial number of failure modes imposes a significant computational burden for massive contingency analysis. It is critical to select a limited set of high-impact contingency cases within the constraint of computing power and time requirements to make it possible for real-time power system vulnerability assessment. In this paper, we present a novel application of parallel betweenness centrality to power grid contingency selection. We cross-validate the proposed method using the model and data of the western US power grid, and implement it on a Cray XMT system – a massively multithreaded architecture – leveraging its advantages for parallel execution of irregular algorithms, such as graph analysis. We achieve a speedup of 55 times (on 64 processors) compared against the single-processor version of the same code running on the Cray XMT. We also compare an OpenMP-based version of the same code running on an HP Superdome shared-memory machine. The performance of the Cray XMT code shows better scalability and resource utilization, and shorter execution time for large-scale power grids. This proposed approach has been evaluated in PNNL's Electricity Infrastructure Operations Center (EIOC). It is expected to provide a quick and efficient solution to massive contingency selection problems to help power grid operators to identify and mitigate potential widespread cascading power grid failures in real time.**

Keywords-powergrid; contingency selection; betweenness centrality; Cray XMT; parallel computing

I. INTRODUCTION

Contingency analysis is a security function to assess the ability of a power grid to sustain various combinations of power grid component failures at energy control centers. The N-1 criterion issued by the North American Electric Reliability Corporation (NERC) is widely used to express the ability of the transmission system to lose a physical component such as a power line or a power generator without causing an overload failure elsewhere. As electricity demand continues to grow and renewable energy increases its penetration in the power grid, multiple contingencies occurring simultaneously across the power transmission network are increasingly likely. The need for N-x ($x \geq 2$) contingency analysis to assess whether the system can withstand the failure of any two or more components becomes necessary.

For large power systems comprising thousands of lines and generators, the contingency analysis process usually imposes a substantial computational burden for real-time operations. For example, in the western US power grid, which includes 17,000 branches, the number of cases could easily reach 1021 for N-5 contingency analysis even if only line outages are considered. It is impractical to examine a combinatorial number of contingency cases, such as these ones. Most of the contingencies have a low probability of occurring and/or have no detrimental consequences on the system stability. Removing them from the rigorous analysis list can dramatically decrease the computational requirement. Contingency selection is thus performed to identify only a limited set of critical contingency cases for further vulnerability assessment and mitigation responses.

Several methods for contingency selection have been proposed since 1979 [1], each varying in methodology and computational complexity [2]. Most of them are based on approximate power flow solutions, and still involve some simplified analysis of all combinatorial contingencies. This makes it computationally impractical for N-x analysis within the constraint of existing computer resources.

In this paper, we propose a new contingency selection method, which is based on applying edge betweenness centrality [3] to power grid topology. By treating the power grid as a weighted undirected graph, we identify high-impact components in the power grid by finding the most "traversed" edges: those with high betweenness centrality indices in the graph. The failures of these selected high-impact components are analyzed to prepare grid operators with mitigation procedures so as to avoid cascading failures. The low-impact components, which are identified as the least "traversed" edges in the graph, are removed from further contingency analysis since their failures are of little importance to the overall power grid stability.

This method has been implemented on the Cray XMT machine, leveraging its advantages for parallel execution of irregular codes, such as betweenness centrality analysis. The performance and scalability of the parallel implementation

is successfully demonstrated and compared with a cache-based, scalable, shared-memory HP Superdome machine.

The paper is organized as follows: Section II presents background materials on the details of the edge betweenness centrality algorithm and the Cray XMT system. Section III presents our algorithmic design and implementation on the Cray XMT. Section IV presents the tests and validation of our contingency selection method with actual impact of power grid component failures. Section V and VI show our experimental results on the Cray XMT and the comparison with the HP Superdome, respectively. Section VII concludes the paper with a brief illustration of the future work.

II. BACKGROUND

We present a short description of the edge betweenness centrality algorithm, as well as the Cray XMT system and its architecture characteristics.

A. Edge Betweenness Centrality

The vertex betweenness centrality of a given vertex in a graph was first defined by Freeman [4] as the fraction of shortest paths, counted over all pairs of vertices that pass through that vertex.

This definition was generalized by Newman and Girvan [3] to edge betweenness centrality, where the betweenness $C_{eB}(e)$ of an edge e is defined as the sum over all pairs of vertices $s,t \in V$, of the fraction of shortest paths between s and t that pass through e (eq. 1). σ_{st} is the number of shortest paths from vertex s to t, and $\sigma_{st}(e)$ is the number of shortest paths from vertex s to t that pass through edge e.

$$C_{eB}(e) = \sum \frac{\sigma_{st}(e)}{\sigma_{st}} \qquad \text{(eq. 1)}$$

Vertices and edges that occur on many shortest paths between any given vertex pair have higher betweenness centrality than those that do not; they have relatively higher importance within the graph. When applied to a real-world network, the vertex and edge betweenness centrality is considered a standard measure of the influence of a node or a linkage over the flow of information between different nodes or links in that network [5].

Edge betweenness centrality has been utilized in numerous studies and implemented in various software tools for network analysis such as biological or social networks [6, 7, 8]. By applying the edge betweenness centrality to the power grid topology, we are able to identify the high-impact transmission lines. An arbitrary number of edges within the top betweenness centrality list can be selected for further N-1 or N-x contingency analysis. To the best of our knowledge this is the first application of edge betweenness centrality to the power grid contingency selection problem.

B. Cray XMT

The Cray XMT supercomputing system [9] is a scalable massively multithreaded platform with global shared memory, which is conducive to large-scale data analysis and data mining.

It incorporates custom Cray multithreaded processors. A single ThreadStorm processor supports 128 hardware thread contexts and is connected with up to 8 GB of memory that is globally accessible by any other processor in the system. The Cray XMT platform can scale to hundreds of thousands of threads. The XMT is a hybrid system, which uses ThreadStorm processors on its main compute partition and AMD Opteron processors for its Service & I/O (SIO) nodes. Software layers enable the use of the x86 SIO nodes for developing hybrid applications.

This massively multithreaded architecture and large global shared memory feature are ideally suitable for parallel applications that require dynamic and random large-scale data access, such as pattern matching, scenario development, behavioral prediction, anomaly identification and graph analysis, which typically do not run well on conventional cluster or supercomputer systems due to lack of locality on many of these algorithms.

The Cray XMT is a suitable platform for our parallel processing of large-scale power grid data. We can use ThreadStorm nodes to perform contingency selection, and use Opteron nodes to perform the floating point computation of the actual contingency analysis.

The Cray XMT programming environment includes advanced programming tools for software development and tuning. Its C/C++ optimizing compiler has an aggressive automatic parallelization capability; programmers can also guide the compiler to parallelize specified loop sections of the code with XMT-specific pragmas.

III. ALGORITHMIC DESIGN AND IMPLEMENTATION

Our algorithmic design is based on a sparse data structure. Extended Dijkstra's algorithm [10] and Brandes's Algorithm [11] are implemented for edge betweenness centrality computation. Hyperlinks in the power grid topology graph are considered in the computation to make the system more representative and comprehensive for contingency selection.

A. Modifed Brandes's Algorithm

The betweenness centrality index is essential in the analysis of weighted networks but is costly to compute. For large and sparse networks, Ulrik Brandes's algorithm [11] is considered to be the fastest algorithm on serial computers [12]. It computes the betweenness centrality in $O(nm + n^2 \log n)$ time, where n and m are the number of vertices and edges in the graph, respectively.

We have adapted Brandes's algorithm to calculate the edge betweenness centrality of the transmission lines (edges) in the weighted power grid graphs, and used an extended Dijkstra's shortest path algorithm [10] to allow it

to store multiple shortest paths between two buses (vertices).

The modified Brandes's algorithm is called n times according to the number of vertices in the graph. In each of the iterations, all the shortest paths between the iterating vertex and all other vertices in the graph are identified, and the edge betweenness centrality value of an edge is incremented whenever an identified shortest path passes through it in this iteration. This strategy is shown in Algorithm 1.

Algorithm 1 Basic steps of the modified Brandes's algorithm for edge betweenness computation

1) Initialize stack, heap and records
2) Put iterating vertex S onto heap
3) While heap not empty:
   a. Remove root v from heap and push it onto stack
   b. For each neighbor w of v, if there exists one or more shortest path(s) from S to w via v:
      i. Set w's distance and number of shortest paths
      ii. Insert or adjust record in heap
      iii. Set/add v as the pre node of w
   c. While stack not empty:
      i. Pop node w off stack
      ii. Increment w's vertex betweenness score
      iii. For each pre node v of w increment v's factor, and edge $v \rightarrow w$'s edge betweenness score
   d. Free the pre node list of w
4) Free records and heap

B. Data Structure

We store the graph as two arrays of structured objects: nodes and edges. A node stores the labels of its neighbors and edges as arrays. The arrays are allocated after the number of neighbors is known, thus no space is wasted. An edge includes a duplicate field used to construct hyperlinks as described in the next section.

The workspace for the modified Brandes's algorithm is an array of records, one for each node. In addition to several real values, a record stores the location of the node in the stack, location of the node in the heap, and the identification number of each predecessor node. Space for one predecessor node is allocated on definition. Space for additional predecessor nodes is allocated and deallocated dynamically. Since a node rarely has more than one predecessor node, dynamically adjusting the size of the predecessor list adds little overhead. The heap is kept as a binary min heap and stored as an array such that the parent of node i is node $(i/2-1)$ and its children are nodes $(2i+1)$ and $(2i+2)$.

C. Hyperlinks

Our application incorporates hyperlinks in the power grid topology graph for the edge betweenness centrality computation.

In power grid systems, the higher the power flowing on a transmission branch, the more important it is for contingency analysis. We use the reciprocal of power flow as the weight of the edge in the power grid topology graph to make the computation consistent with this physical concept. Edges with shorter distances, as determined by the smaller reciprocal power flow value, have a greater chance to lie on multiple shortest paths, and thus have higher betweenness centrality values for contingency selection.

There may be multiple transmission lines between two buses to transmit power in the transmission network. To evaluate the importance of each individual line between two buses, we first compute the betweenness centrality of the hyperlink, which represents the transmission property between the two buses, with a single weight comprising the weights of all edges based on the parallel connection principle as shown in eq. 2,

$$W_{hyper} = 1/(\sum_{i=1}^{n} \frac{1}{W_i}) \qquad \text{(eq. 2)}$$

where n is the number of multiple edges, W_i is the reciprocal of power flow on edge i, and W_{hyper} is the weight for the hyperlink. After computing the edge betweenness score of each hyperlink, we divide the score among component edges according to their weight contribution to the hyperlink.

We construct hyperlinks and compute their weights in three steps. First, we sort the edges using a two key radix sort {head, tail}, and set the duplicate field of each edge to -1. Second, for each edge e_{i+1} that has the same head and tail node as e_i, we set the duplicate field of e_{i+1} to i creating a linked list of edges for each hyperlink. Third, the first edge of each hyperlink (those edges whose duplicate field is -1) computes the weight of the hyperlink storing the value in its weight field. Note, the original weight of the first edge is overwritten, but it is easily recomputed from the total weight and weight of the other component edges.

After constructing the hyperlinks, we construct the neighbor list of each node storing the index of the first edge of a hyperlink as the label of the edge between two neighbors. It is worth mentioning that if power transmission corridors instead of parallel lines should be considered in contingency analysis, the hyperlinks can be collapsed to a single link using equivalence of parallel lines.

D. Parallel Implementation on Cray XMT

The Cray XMT system has many advantages for parallel graph-processing applications. Our codes are written in a way that facilitates the automatic parallelization by the XMT compiler. We also use XMT-specific pragmas to guide the compiler when automatic parallelization is not possible [13].

When iterating through all the vertices to compute the edge betweenness centralities as shown in Algorithm 1, each of the iterations is independent from each other since they

analyze independently-sourced shortest paths in the graph. We add the semantic assertion *"#pragma mta assert parallel"* before this outer loop construct to assert that the separate iterations of this loop may execute concurrently without synchronization. We also add the compilation directive *"#pragma mta use 100 streams"* to ensure that a sufficient number of resources are used to maximize utilization.

The steps to compute hyperlinks comprise a series of parallel loops. The compiler automatically parallelizes several of the loops. For those that it does not parallelize, we insert the directive *"#pragma mta assert no dependence"* to force parallization. In several places in the code, we use the atomic increment function *int_fetch_add(&v,i)* to atomically add i to the value at address v, and return the original value.

To atomically update floating point values, we insert the directive *"#pragma mta update"* before the corresponding source code statements; for example

$$Nodes[node].vb+ = records[node].data;$$

and

$$Edges[edge].vb+ = factor;$$

IV. VALIDATION

We have validated our edge betweenness based contingency selection method considering a practical power grid perspective.

We use General Electric's Positive Sequence Load Flow Software (PSLF) – a widely used commercial grade power system analysis tool – to perform power grid contingency analysis. For each contingency case, a performance index (PI) [1] is calculated and all indices are ranked based on eq. 3,

$$PI_i = \sum_{j=1}^{n-1} \left(\frac{P_j}{P_{j\max}} \right)^2 \qquad \text{(eq. 3)}$$

where n is the number of transmission branches (edges), P_j is the power flow in each transmission branch (edge), and $P_{j\max}$ is the power capacity for each transmission branch (edge).

A power system model containing 760 buses (vertices) and 977 branches (edges) extracted from an area of the western US power grid (WECC) is used to test and validate the proposed contingency selection method with actual impact. To be consistent with the PI definition, the obtained edge betweenness is reweighted by the power capacity $P_{j\max}$ of the edge.

The distribution of edge betweenness scores for the 760-bus power system is shown in Figure 1. We can see that there is a small portion of edges that have high edge betweenness scores, which means that this small portion of edges is important to this power grid graph. The same idea holds true in the real power system: there is a small set of transmission lines that are more critical compared with other lines.

A representative cross-validation result is shown in Figure 2, where the x-axis is the percentage of cases selected from contingency ranking results based on PIs, and the y-axis is the percentage of common cases between PI-based selection and edge-betweenness-based selection. From Figure 2, we can see that with 70% contingency cases selected (x-axis), the hit rate using edge betweenness can achieve 70% (y-axis). This high hit rate demonstrates the validity of the proposed edge-betweenness-based method. It can provide a practical list of critical lines in the power grid system for contingency analysis. Validation cases with power systems of various sizes exhibit similar performance.

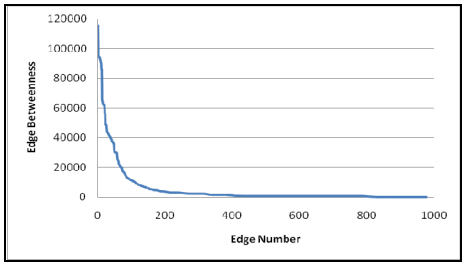

Figure 1. Edge betweenness distribution for the 760-bus system.

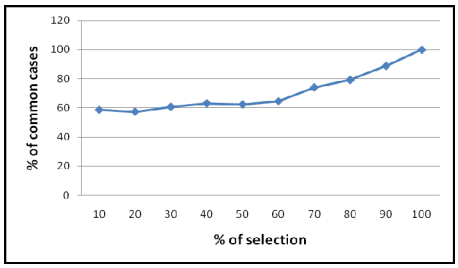

Figure 2. The cross-validation results for the 760-bus system.

V. PERFORMANCE ANALYSIS

We have implemented the edge betweenness centrality parallel algorithm as described in Section III on the Cray XMT multithreaded system. The configuration of the system is:

- 64 500MHz 64-bit Cray ThreadStorm Processors
- 128 Threads per processor
- 512 GB of global shared memory

We also leverage power grid resources available in our Electricity Infrastructure Operations Center (EIOC) and perform the analysis using real-world models and data.

A. Study Cases

To assess the computational performance of the edge betweenness centrality application for power grid contingency selection, we select four power system models of different sizes as task graphs as shown in Table I.

TABLE I. Study Cases

Name	# of Vertices	# of Edges
14,000-bus	14,090	17,346
46,000-bus	46,010	57,323
90,000-bus	92,370	116,980
170,000-bus	172,930	220,648

The 14,000-bus represents the western US power grid (WECC). The 46,000-bus system, 90,000-bus system, and 170,000-bus system are extensions of WECC by adding more detailed power networks at sub-transmission and distribution levels, which represent potential capabilities to analyze power systems in more detail. With more and more renewable generation and other distributed generation penetrating distribution systems, the need to expand the modeling coverage is a compelling need.

B. *Resource Utilization*

The processor utilization of Cray XMT is a measure of the effectiveness of memory access latency hiding. It is obtained by measuring the number of clock cycles in which instructions have been issued versus the total number of clock cycles spent in the computation.

We have measured the memory usage and processor utilization for all study cases. In all of these cases, we observe a relatively stable high processor utilization rate (~30%), which indicates the effectiveness of the multithreaded capabilities of the ThreadStorm processor in covering the latencies of the irregular memory accesses required to traverse the graph for betweenness centrality computation. Another observation is the proportionally increased memory usage with the increased graph size and number of processors used. In Table II, we present the memory usage and utilization rate for 14,000-bus study case when running with different number of processors on the Cray XMT. It is important to note that the memory usage is proportional to the number of active threads in the system, since each thread keeps a private stack, heap and state variables as described in Algorithm 1. Thus using more processors increases more memory needed for these variables.

TABLE II. Memory Usage and Utilization Rate versus the Number of Processors of the 14,000-bus System

# of Processors	Memory Usage (GB)	Utilization Rate (%)
1	0.455	30
2	0.723	30.4
4	1.3	30.5
8	2.2	30.3
16	4.9	29.4
32	8.5	26.7
64	15.9	26.1

C. *Scalability*

To measure the scalability of the edge betweenness implementation on the Cray XMT machine, we record the computer execution times for the four study cases against different number of processors. Figure 3 shows the execution time on different numbers of processors. Figure 4 shows the corresponding speed-up curve for each case. Both figures use a logarithmic y-axis to improve the readability of the graph.

Figure 4 indicates that the scalability of the code is close to linear for all problem sizes, with a trend of better scalability for larger problems. These results indicate that the Cray XMT is very suitable for the proposed betweenness centrality analysis method operating on large power grid graphs.

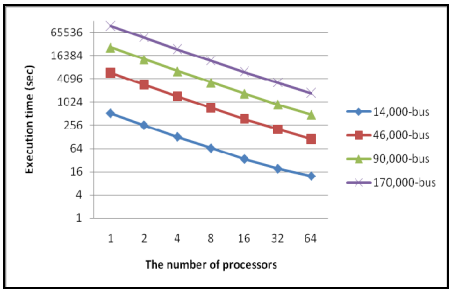

Figure 3. Execution time versus the number of processors for power graphs of various sizes on the Cray XMT.

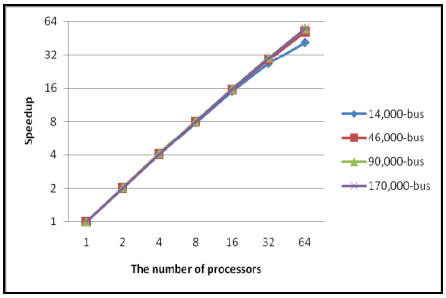

Figure 4. The speed-up curve for power graphs of various sizes on the Cray XMT.

VI. Comparison Between Cray XMT and Superdome

We have also built an OpenMP-based version of our application that utilizes the same algorithm as the Cray XMT version, as a way of comparing a scalable shared-memory machine (HP Superdome) against the XMT.

The configuration of the Hewlett-Packard Superdome system is:

- 64 1.6 GHZ dual-core 64-bit Itanium 2 processors
- 24MB of L3 cache per processor
- 256 GB of global shared RAM

The same four cases are used to test the performance of the Superdome machine. Figure 5 and Figure 6 show the execution times and the speedup numbers, respectively.

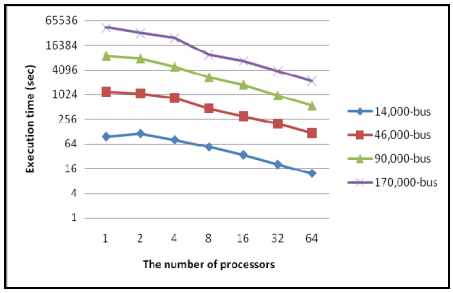

Figure 5. Execution time versus the number of processors for power graphs of various sizes on the Superdome.

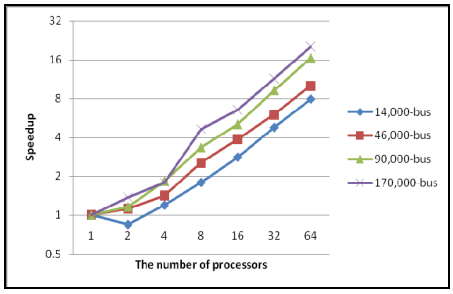

Figure 6. The speed-up curve for power graphs of various sizes on the Superdome.

Table III lists the execution times for the four study cases on the Cray XMT and the Superdome when different numbers of processors are used. Figure 7 compares their relative execution time in plots. The data presented in Figure 7 has been normalized to the execution time on one processor on the Superdome.

Table IV lists the speedup numbers for each case. The speedup numbers for both Cray XMT and HP Superdome are calculated by normalizing the execution time using their respective single processor time. Figure 8 compares their speedups in plots. Again, logarithmic y-axis is used in both of the figures to improve the readability of the graphs.

TABLE III. EXECUTION TIME (SEC) VERSUS THE NUMBER OF PROCESSORS FOR FOUR STUDIES ON THE CRAY XMT (C) AND THE SUPERDOME (S)

Study cases	C/S	1p (sec)	2p (sec)	4p (sec)	8p (sec)	16p (sec)	32p (sec)	64p (sec)
14,000 -bus	C	515.23	254.88	127.53	65.35	34.2	19.38	12.45
	S	95.16	112.64	79.83	53.27	33.91	19.96	12.02
46,000 -bus	C	5884.2	2903.9	1442.6	732.46	378.05	202.13	112.85
	S	1179.7	1054.3	831.22	463.61	303.83	195.79	116.85
90,000 -bus	C	26934	13179	6510	3298.9	1703.8	896.08	486.42
	S	8976.5	7747.1	4893.6	2688.5	1776.2	970.06	542.78
170,00 0-bus	C	96680	47822	23811	12020	6205.1	3267.5	1745.1
	S	44088	32149	24505	9464.5	6738.1	3836.3	2166.9

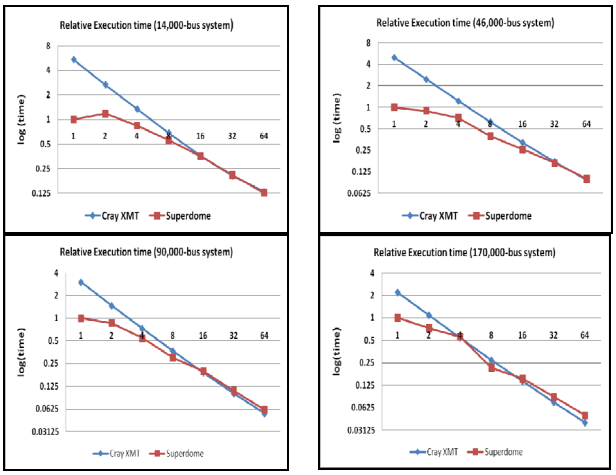

Figure 7. Comparison of Cray XMT and Superdome relative execution time for four different problem sizes.

TABLE IV. SPEEDUP NUMBERS VERSUS THE NUMBER OF PROCESSORS FOR FOUR STUDIES ON THE CRAY XMT (C) AND THE SUPERDOME (S)

Study cases	C/S	1p	2p	4p	8p	16p	32p	64p
14,000 -bus	C	1	2.021	4.04	7.884	15.065	26.654	41.384
	S	1	0.845	1.192	1.786	2.806	4.768	7.917
46,000 -bus	C	1	2.026	4.079	8.034	15.565	29.111	52.142
	S	1	1.119	1.419	2.545	3.883	6.025	10.096
90,000 -bus	C	1	2.044	4.137	8.165	15.808	30.058	55.372
	S	1	1.159	1.834	3.339	5.054	9.254	16.538
170,00 0-bus	C	1	2.022	4.06	8.045	15.581	29.588	55.401
	S	1	1.371	1.799	4.658	6.543	11.492	20.346

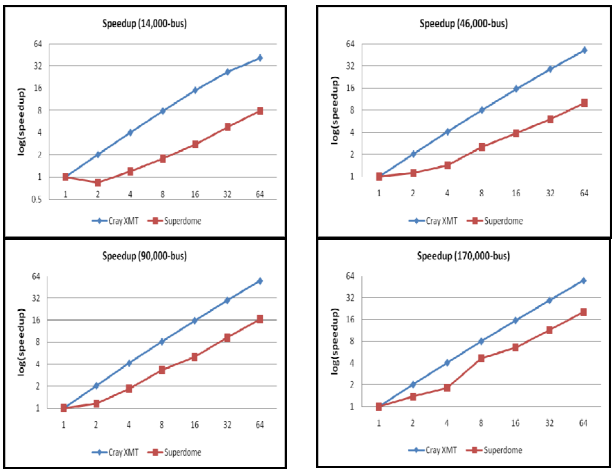

Figure 8. Comparison of Cray XMT and Superdome Speedup for 4 different problem sizes.

Comparison of Figure 7 and Figure 8 indicates that the Cray XMT machine achieves superior scalability for graph edge betweenness computation over the Superdome machine.

The hardware and software features of the Cray XMT are a better fit than the Superdome platform for graph

computation. Its global shared memory, very fine-grained threading, and efficient light-weight word-level synchronization makes it well-suited for parallel graph processing. We can achieve up to 55x speedup against the one-processor version of the code using 64 processors on the Cray XMT for the 170,000-bus system.

For smaller graphs, the Superdome can take advantage of its large caches to store the data and thus dramatically reduce the average memory access time. But when the graph is larger and data fetching between different computational nodes is frequent, the advantage of the caches diminishes, and the overall computational performance degrades due to inter-node communication. As shown in the graphs, when 64 processors are used on the Superdome, only 20x speedup is achieved for the 170,000-bus system.

VII. CONCLUSIONS

Contingency selection is a key step for real-time power system contingency analysis. In this paper, we present a novel application of edge betweenness centrality to conduct the contingency selection for power grid topology graphs. The parallel implementation of this method on the Cray XMT machine exhibits good performance in absolute execution time, as well as good scalability with increased problem sizes. It demonstrates that the Cray XMT is well-suited machine for our graph processing in contingency selection.

In the future, we will take advantage of the Cray XMT's hybrid capabilities with ThreadStorm and Opteron nodes. Further work will focus on the communication between ThreadStorm nodes, which performs the graph-based contingency selection, and Opteron nodes, which will be used for floating-point computation of actual contingency analysis. We will further evaluate this method and port the developed software in the Electricity Infrastructure Center at the Pacific Northwest National Laboratory. It is expected to provide a more efficient solution to the N-1 to N-x contingency problems to help the operators to identify and mitigate potential widespread cascading power grid failures in real-time power system operations.

ACKNOWLEDGEMENTS

This work is supported by the Center for Adaptive Supercomputing Software – Multi-Threaded Architectures (CASS-MT) funded by the Department of Defense and by the Electricity Infrastructure Operations Initiative of the Pacific Northwest National Laboratory. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC06-76RL01830. Acknowledgement is extended to David Haglin, and Douglas Baxter, both with the Pacific Northwest National Laboratory, for productive discussions and suggestions.

REFERENCES

[1] G. C. Ejebe, and B. F. Wollenberg, "Automatic contingency selection," IEEE Trans. on Power Apparatus and Systems, vol. PAS-98, No. 1, pp. 92-104, Jan./Feb. 1979.

[2] A. O. Ekwue, "A review of automatic contingency selection algorithms for on-line security analysis," IEEE Conf. Publ, v336, pp. 152-155, 1991.

[3] M. Girvan, and M. E. Newman, "Community structure in social and biological networks," Proc. Natl. Acad. Sci. USA, 99, 7821-7826, 2002.

[4] L. C. Freeman, "A set of measures of centrality based on betweenness," Sociometry, 40, 35-41, 1977.

[5] W. P. John, and R. W. David, "Betweenness-based decomposition methods for social and biological networks," Interdisciplinary Staticsa and Bioinformatics Leeds University Press, pp. 87-90, 2007.

[6] S. Y. Chan, I. X. Y. Leung, and P. LiÒ, "Fast centrality approxmiation in modular networks," Conference on Information and Knowledge Management, Proceeding of the 1st ACM international workshop on Complex networks meet information & knowledge management, pp. 31-38, 2009.

[7] D. A. Bader, and K. Madduri, "Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks," Proc. 35th International Conference on Parallel Processing (ICPP), Columbus, OH, August 2006.

[8] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. G. Chavarría-Miranda, "A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets, " Third Workshop on Multithreaded Architectures and Applications (MTAAP), Rome, Italy, May 29, 2009.

[9] Cray XMT Supercomputer Brochure, http://www.cray.com/products/XMT.aspx.

[10] E. W. Dijkstra, "A note on two problems inconnexion with graphs," Numerische Mathematik, 1, S. 269-271, 1959.

[11] U. Brandes, "A faster algorithm for betweenness centrality," Journal of Mathematical Sociology, Vol. 25, pp. 163-199, 2001.

[12] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail, "Approximating betweenness centrality," The 5th Workshop on Algorithms and Models for the Web-Graph (WAW2007), San Diego, CA, December 11-12, 2007.

[13] Cray XMT™ Programming Environment User's Guide.