Power Distribution Management System revisited: Single-thread vs. Multithread Performance

Alma Husagic-Selman

aselman@ius.edu.ba, International University of Sarajevo

Article Info

Article history:
Received 17 Jul. 2015
Received in revised form 17 Aug. 2015

Keywords:
Power Distribution Management System, multi processor, multithread processing, Real-time, scheduling, closed loop, smart grid

Abstract

Power Distribution Management System (PDMS) uses very sophisticated algorithms to deliver reliable and efficient functioning of power distribution networks (PDN). PDNs are represented using very large sparse matrices, whose processing is computationally very demanding. Dividing large PDNs into smaller sub-networks results in smaller sparse matrices, and further processing each sub-network in parallel significantly improves the performance of PDMS. Using multithreading to further process each sub-network however degrades PDMS performance. Single-thread processing of sub-network sparse matrices gives much better performance results, mainly due to the structure of these matrices (indefinite and very sparse) and synchronization overhead involved in multi-thread operations. In this paper an overview of PDMS system is presented, and its performance given single-thread and multiple threads is compared. The results have shown that for some applications, single-threaded implementation in multi-process parallel environment gives better performance than multithreaded implementation.

1. INTRODUCTION

Modern power distribution networks are huge and complex, and they require well designed tools for reliable system supervision and control, and fast data processing [1]. As the networks grow, their computational time also grows, but system requirements demand the computational time to decrease. To achieve that, software modifications and improvements of network analysis and management tools are essential.

Most of the electric utilities in PDNs around the world have installed numerous SCADA (Supervisory Control and Data Acquisition) systems on their substations and feeders and in combination with PDMS, they ease the process of real-time data acquisition, system management and control [2]. These systems contain the state estimator and load flow programs and are used to pre-process data and execute some calculations whose results are used in subsequent data analysis [1].

Nevertheless, many implementations of network applications are still based on sequentially working programs. This was fine in the present years, as most of the computers only have a single CPU, which executed parallel processes taking advantage of time multiplexing of the operation system.

With the emergence of multi-core processors, the processing power of computational devices has increased tremendously. Today, computers with several CPU-cores are in common use, and in modern computer clusters, the number of CPU cores can be significantly larger [1]. As result, parallel processing is becoming a trend in applications that require intensive computations.

Even with increased processing capabilities of multi-core CPUs, it is very challenging to improve
computational performance of PDN systems. Main reason for this is difficulty in adapting existing system algorithms to parallel computing features of the devices. Often, complete system application might need to be carefully redesigned to fully utilize the hardware architecture [3].

In the past, many different publications addressed parallel execution issues of network applications, considering system architecture, software and hardware.

Authors in [4-8] show multiple agent technology and some other architectural issues related to power system applications. Multi-agent technologies have found applications in many distributed systems such as distributed problem solving, distributed information fusion, and distributed scientific computing [5, 9].

[10-15] discussed the software to be used for PDN problems. Software package OpenMP makes the parallelizing of software much easier. This package allows defining parts of the software to be executed in parallel. It helps parallelize the software application code. Using only compiler directives can speed up the execution significantly [10-14]. OpenMP uses threads for the execution of sub-processes. The organization of them is handled in the OpenMP package itself, using standards of the operation system. With OpenCL, another package exists, which also uses the internal cache of modern computers for fast calculations [16].

Important discussion topic in last two decades is hardware architecture and its impact on computational speedup. SIMD (single instructions multiple data) architecture allows loading of the processor’s register in an optimized manner, resulting in execution of several instructions at the same time. [15, 17-18]. SIMD execution can be implemented on both, multi-core CPU (using SSE and AVX instructions) and GPU (Graphic Processing Unit) processors.

Multi-thread processing is designed to maximize the utilization of multi-core or multiple processor devices [15-17, 21, 22]. Applications that involve sparse matrices, use multithreading to boost computational performance of the algorithms involved [21, 22]. In systems with shared memory, computational performance deteriorates as the number of processing elements is increased [27, 28].

This paper presents a review of already implemented real-life PDMS, and compares its performance given multiple threads and a single. In this system, large PDN is divided into sub-networks, and each sub-network is passed to a distinguished processor core, for further processing either via single or multiple threads. The computation times are presented and discussed at the end of paper.

2. DESCRIPTION OF THE PDMS

Power control centre is used to monitor and control PDN. It consists of multiple entities, such as SCADA, GIS, database, user interface and PDMS.

SCADA module receives real time measurements from all monitored substations via one of many possible types of communication links. These measurements, which are in raw form, are passed to PDMS for further processing. After the data is processed SCADA receives the actions to be taken from PDMS.

GIS (Geographical Information System) captures, stores, manipulates and manages all types of geographical data. This module sends map data of PDN to the PDMS, to get the map coloring the information returned from the system. The mapped with exact location is then forwarded to the personnel in charge.

PDMS (Power Distribution Management System) is a set of tools used to supervise and control power distribution network. It is essential part of Power Control Centre, as it performs the estimation, control, optimization, fault detection and restoration of distribution network. The most important functionalities of PDMS are:

1. **Supervisory control and data acquisition (SCADA)**: provides interface to the real-time data [29]-[33].
2. **Distribution System State Estimation (DSSE)**: estimates the power flow using actual measurements from the technical process. This is the most important part of PDMS [34]-[48].
3. **Distribution System Power Flow (DSPF)**: based on the known load profile, this block computes power flow for research and simulation [49]-[60].
4. **Voltage-Var Control (VVC)**: controls and changes controllable network equipment (LTC transformers, capacitors, distributed generators, etc.) in order to optimize network state. In other words, it optimizes power flow by changing some control parameters of the network [61]-[79].
5. **Short Circuit Calculation (SCC)**: detects faulty areas of the network. This is used as an input to the Fault Location and Service Restoration application suite [80]-[84].
6. Fault Location and Service Restoration: this application suite is used to isolate fault within the network and restores the power supply for non-faulty areas [85]-[89].

7. Optimal Feeder Network Reconfiguration (OFR): optimizes network by reconfiguring its topology, especially in the case of feeder overload. Switching procedure transfer loads from one feeder to other and in that way reconfigure the network [90]-[95]. This is very important application for distribution networks. It is responsible for setting switch states in such manner that network operates in safe and economically optimal manner.

8. Graphical User Interface: Data is visualized using single-line diagrams presented in very usable Graphical User Interface [96]-[103].

3. NETWORK SUB-DIVISION AND PROCESS SCHEDULING IN PDMS

Power distribution networks are expressed as sparse matrices, and processing large sparse matrices is computationally very demanding process. The larger the network, the larger the matrix, and more time is needed for its computation. With network of up to 1,000,000 nodes, it is hardly impossible to calculate the complete network in one step. Luckily the structural features of a typical distribution network allow its splitting into smaller sized independent sub-networks (Figure 2) [1].

Dividing the problem into independent sub-problems simplifies and accelerates computations, as the solution of the complete network is now split to a combination of solutions of many smaller sub-networks. This reduces the memory consumption and also enables parallel processing of sub-networks. Each sub-network can now be fed into a separate processor core, or even separate computers.

The whole concept functions in the following way: PDMS system receives the network data from SCADA. The network gets divided into smaller sub-networks, and sub-networks get passed to respective queue in PDMS modules (DSSE, PF, VVC, OFR, etc.) (Figure 3).

Each module has its own scheduling mechanism (scheduler), with a number of allocated slave processes. The number of processes per scheduler depends on hardware configuration. Scheduler maintains the queue of sub-networks, and it functions on the basis of First-Come-First-Serve (FCFS) mechanism. The role of the scheduler is to ensure that each sub-network from the queue gets to processing as soon as at least one process is free. As soon as the slave process receives the sub-network, the scheduler locks that network and no configuration change can be done until its processing is complete. Figure 4 demonstrates how scheduling mechanism dispatches the sub-networks to slave processes. Whenever the calculation of a sub-network is completed, freed slave process gets assigned next sub-network from the queue.

In the case that the system allows triggering of events, the triggered sub-network will have its priority increased, and it would get to processing as soon as one slave process is freed. This is demonstrated in Figure 5.
Because of sub-network independence, there is no need for synchronization between different slave processes. Synchronization is only needed for scheduler-slave communication and for the case the sub-network gets further divided after its computations has started. If new sub-networks processes, it’s very probable that they will finish computation before the original larger sub-network. Upon finishing, obsolete sub-network would overwrite results from smaller sub-networks. The scheduler solves this problem by postponing the calculation start of the new sub-networks for several seconds. This small delay is acceptable, as any other solution would complicate scheduler design and introduce other synchronization problems (additional communication between master-slave processes) and consequent delays.

4. PDMS COMPUTATIONAL MODULES SOFTWARE DESIGN APPROACH

In multi-core environments, avoiding idle processor cores during the computation is one of the key performance aspects. Increased level of parallelism should be good for the system, as parallelism increases the processor’s utilization and boosts overall system’s performance. The largest limitation in such systems is the memory: the ratio of peak memory bandwidth to peak floating-point performance (byte : flop ratio) is decreasing as core counts increase [27]. In other words, larger the number of cores in the system, smaller the memory bandwidth per core.

In PDN, data obtained from SCADA is in form of large matrices that are irregular, indefinite and very sparse. They are the sparsest matrices encountered in real life applications. Due to the matrix sparsity, memory access patterns are irregular and utilization of caches suffers from low spatial and temporal locality [28]. These systems in general have large number of cache misses and their performance is therefore bounded by main memory speed.

Sparse matrix calculations are iterative – they involve repetition of a single operation on row/column sets of distinguished data. This makes them convenient for parallelism using multithreading. With parallelism added, sparse matrix computation gets divided among multiple threads. For example, if one thread starts computing first row of the matrix, and the other fifth, the system will require the data of the first and fifth row to be pre-fetched to the cache memory. Since it is not possible to fetch all data for both threads, and because of the low spatial locality of data, large number of cache misses would occur, and performance of the system would be bad.

PDMS system presented in previous section already implements parallelism by having different sub-networks run on multiple slave processes, with each process being executed on separate core. Adding additional parallelism by further dividing sub-network computation and assigning these independent computation tasks to multiple threads would only cause performance degradation. This is because each thread would request its share in processor cycle and cache memory. Ultimately the threads would be stalling the processor while waiting for data due to inevitable cache miss (thread spin).

It is important to note that, aside from the PDMS application, computer system has other application running via multiple threads at the same time, all of them expect their share in memory and processor cycle. Further increase in the level of parallelism would inevitably cause performance stall.

Single processor cores achieve peak performance if they execute straight serial codes without conditional branches [28]. When designing computational system that will run on multi-core processors, it would be recommended to consider this fact. As sparse matrix computations involve iterations (conditional branches), the peak performance cannot be achieved with current algorithms, some cache misses will always occur due to matrix structure, but executing the computations serially on individual cores would definitely make the performance improvements. This is proven in following section, where the comparison of single-threaded and multithreaded performance is presented.

5. RESULTS

Parallel processing of a single sub-network was tested using a single thread and multiple threads. Test network was three-phase non-symmetrically balanced, and it consisted of 620 different sub-networks. Network equipment units are summarized in Table 1.

The hardware used during the testing had the following features:
- Processor: 2x Intel Xenon
- Total number of processor cores: $2 \times 4 = 8$ cores

To test the timing, Win32 API timing functions are used directly in code. Minimal logging is used and no compiler optimization was done on code.
Table 1: Number of equipment units in test network

<table>
<thead>
<tr>
<th>Type of Equipment</th>
<th>No. of equipment units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>1,227,600</td>
</tr>
<tr>
<td>Transformers</td>
<td>75,020</td>
</tr>
<tr>
<td>Lines</td>
<td>1,205,280</td>
</tr>
<tr>
<td>Loads</td>
<td>639,840</td>
</tr>
<tr>
<td>Capacitors</td>
<td>66,960</td>
</tr>
<tr>
<td>Generators</td>
<td>620</td>
</tr>
<tr>
<td>Disconnectors</td>
<td>949,840</td>
</tr>
<tr>
<td>Ground Disconnectors</td>
<td>342,240</td>
</tr>
<tr>
<td>Breakers</td>
<td>359,600</td>
</tr>
<tr>
<td>Load Break Switches</td>
<td>1,236,280</td>
</tr>
<tr>
<td>Measurements</td>
<td>1,010,600</td>
</tr>
<tr>
<td>Voltage Controllers</td>
<td>5,850</td>
</tr>
</tbody>
</table>

Table 2: Comparison results for running of multiple DSSE single-threaded processes

<table>
<thead>
<tr>
<th>Test / Number of processes</th>
<th>1[s]</th>
<th>2[s]</th>
<th>4[s]</th>
<th>8[s]</th>
<th>8+[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodical run all sub-nets</td>
<td>2791,5</td>
<td>1691</td>
<td>1138</td>
<td>933</td>
<td>830</td>
</tr>
<tr>
<td>Average per Sub-Net</td>
<td>4,50</td>
<td>2,73</td>
<td>1,84</td>
<td>1,51</td>
<td>1,34</td>
</tr>
</tbody>
</table>

Table 2 presents measurement results in seconds. Periodical run is measured when all of the sub-networks are calculated in one run, and average time per sub-network is the average execution time of individual sub-networks. Figure 6 and 7 show the plotted results respectively. Figure 8 shows the performance of BLAS under the same conditions. It can be seen that the presented model outperforms the BLAS by the factor of 20.

Table 3: Comparison results for running of multiple DSSE multi-threaded processes on 8-core workstation

<table>
<thead>
<tr>
<th>Test / Number of threads per application</th>
<th>2[s]</th>
<th>4[s]</th>
<th>8[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodical run all sub-nets</td>
<td>2418,27</td>
<td>3759,33</td>
<td>4854,42</td>
</tr>
<tr>
<td>Average per Sub-Net</td>
<td>4,30</td>
<td>6,13</td>
<td>8.62</td>
</tr>
</tbody>
</table>

When multithreading is used, the performance is dreadfully decreased. Table 3 presents measurement results for the same network with four threads per processor core. It is obvious that, in real-time environment where significant number of applications tries to access shared memory simultaneously, the shared memory is becoming performance bottle-neck. Performance comparison of single thread and 4-thread systems is shown in Figure 9.
Further performance improvement can be achieved by assigning processes to exactly specific cores and avoiding operating system load balancing. Even in this case, multithreading in setup as specified in this work would not be a good option.

5. CONCLUSION

Power distribution networks require well-designed tools that maximize hardware utilization in order to provide responsive, robust and reliable functioning of the network. In this paper we gave an overview of how such system is implemented in the real world. We also presented the performance of such system in different processing setups.

As the real power distribution networks are large, their processing takes long time. PDN features allow it to be divided into smaller, independent sub-networks, whose processing can be done in parallel on today’s common multi-core processors. Processing of each sub-network, however should be done using a single thread, because multi-thread processing within a single core degrades the performance of such system. With the trends of growing number of cores, heterogeneous processor core architecture, and newest GPU/CPU integration, the presented results can be used as a basis for further research in this field.

REFERENCES


