# A Multi-core High Performance Computing Framework for Distribution Power Flow

*Tao Cui and Franz Franchetti*
Department of Electrical and Computer Engineering
Carnegie Mellon Univeristy
5000 Forbes Ave. Pittsburgh, PA 15213
Email: {tcui,franzf}@ece.cmu.edu

***Abstract*—There is an enormous growth in performance capability of computing platform in the last decade. The parallelism becomes an inevitable trend for future computing hardware / software design. Motivated by the practical computation performance demands in power system, especially distribution system, and the advances in modern computing platform, we developed a high performance parallel distribution power flow solver for Monte Carlo styled application. From computer architecture and programming point of view, we show that by applying various performance tuning techniques and parallelization, our distribution power flow solver is able to achieve 50% of a CPU's theoretical peak performance. That is 50x speedup comparing to an already fully compiler-optimized C++ implementation.**

## I. Introduction

Power flow computation is the most essential routine for power system analysis. It often lies on the critical path of most power system analysis and simulation programs. Distribution power flow is a computation model and method specified for distribution system which often has multi-phase unbalanced parameters, high R/X ratio and radial structure [1] [2] [3]. With the recent development of smart grid technologies and integration of renewable energy resources into distribution system, there are more and more performance demands and research efforts on power flow computations in distribution system [3] [4] [5]. One compelling application case is applying probabilistic power flow for distribution system to deal with the uncertainties of renewable energy resources. Since the renewable energy is stochastic in nature, the deterministic power flow results become inefficient in solving the system's states. Lots of on-going researches are trying to apply probabilistic load flow for distribution system by modeling the renewable energy resources as random variables or stochastic processes [6] [7] [8] [9]. Among the probabilistic power flow methods, Monte Carlo simulation is one of the numeric solution that often serves as the "gold standard" for accuracy reference. However, Monte Carlo styled methods often suffer from the high computation burden. Previously, such method can only serve as off-line program to evaluate the accuracy of other probabilistic power flow methods, and often high performance supercomputer has to be employed for Monte Carlo simulation.

This work was supported by NSF through awards 0931978 and 0702386.

Meanwhile, the performance capability of computing platform has been growing rapidly in last several decades. Moore's law would still be a good prediction of the trend of computing industry [10]. Nowadays, the mainstream CPU or GPU accelerated systems enable us to build a personal supercomputer at a very low cost, which may have the similar computation power comparing to the fastest supercomputers in the world just less than ten years ago. However, due to the power wall of hardware design, from the beginning of this century, the rapid increase of performance is more achieved by shifting from increase of clock frequency to parallelization in hardware / software models. This means extracting these computation power from the hardware is not easy any more. Parallel programming model has to be applied to take advantages of the hardware advances. Besides, other hardware constraints such as memory hierarchy have to be considered for performance tuning in order to fully extract the computation capacity for specified numerical applications' performance [11].

In this paper, we are standing in the middle of computational performance demand of power system application and the rapid growing and model shifting of modern computing platform. We are looking into the Monte Carlo styled applications in distribution system analysis. From the software model point of view, the high computation burden Monte Carlo simulation is actually a favorable case as it might be easy parallelizable. We applied data parallelism and multi-threading programming as well as the performance tuning techniques for dynamic data structure and memory hierarchy. The core computation is to solve many power flows efficiently and simultaneously. The goal is to squeezing the computation power out of the modern computer architecture, push the application's performance to the hardware peak limit. After parallelization and other optimization we are able to achieve 50% of the theoretical peak performance of a CPU, that is 50x speedup comparing to already compiler fully optimized baseline C++ implementation on a single CPU desktop system, and on a two-CPU server system it is about 150x speedup. The result has show that, without extra work on performance tuning and parallel programming, the specified power flow application's performance can suffer, about 99% of the designed hardware computation power might be wasted. By applying parallel programming models and proper performance tuning tech-

niques, the modern mainstream desktop computing platform can yield a similar high performance result comparing to the supercomputers several years ago. This result can also be regarded as a demonstration case that shows how conventional power grid computation and analysis can benefit from the rapid development of computing hardware/software platform.

## II. Advances and Challenges in Computation

The performance capabilities of computing platforms have been growing rapidly in last several decades at a roughly exponential rate. Currently a mainstream Intel server CPU can deliver a floating point operation speed of almost 200 Gflops, that is $2 \times 10^{11}$ additions/substractions/mulitplications per second. This is only one single CPU chip, another 10x increase can be achieved with graphics accelerator and multiple CPUs. To better interpret the speed, we compare the peak performance of commercial off the shelf (COTS) Intel CPU with several fastest supercomputing system in the world (on Top500 list [12]) ten years ago in the following table I.

TABLE I
Theoretical Peak Performance Comparison

| Systems / Chips | Peak Gflops | Year | Position |
|---|---|---|---|
| SP Power3 375MHz | 78.0 | 2000 | Top 500 |
| T2E1200 | 139.2 | 2001 | Top 500 |
| SuperDome/HyperPlex | 281.6 | 2002 | Top 500 |
| Intel i7-975 | 105.0 | 2010 | COTS Desktop |
| Intel Xeon X5680 | 162.0 | 2010 | COTS Server |

Table I shows the peak performance of 500-th fastest supercomputer in the world on Top500 list about ten years ago and the peak performance of COTS Intel desktop/server CPU [13]. Clearly, the Intel CPUs released in 2010 have similar, even higher theoretical peak performance comparing to the world's top 500 fastest supercomputers about ten years ago. We are using theoretical peak performance value as "Peak Gflops". The performance data may give us an rough implication that certain numerical application which can only run on supercomputer ten years ago may be possible to just fit into a single desktop personal computer nowadays.

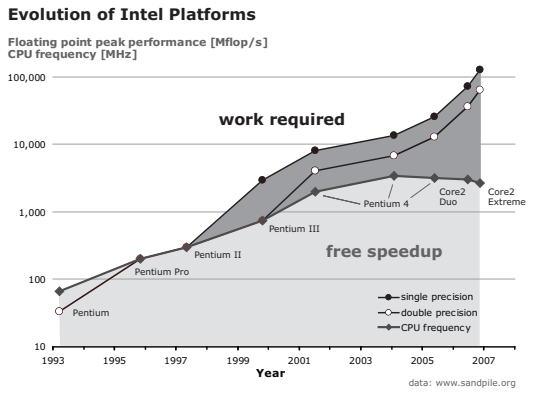


Fig. 1. The evolution of Intel CPU peak performance v.s. CPU frequency [11]

However, to extract such computation power out of the hardware architecture is not easy.

As we can observe from the Fig. 1. Before the end of last century, the CPU clock frequency is growing at a roughly exponential rate. The clock frequency increase roughly means a "free speedup" for existing numerical computation software, but even in those times, the increasing of CPU clock frequency has caused the bottleneck between processor and memory systems, the code has to be optimized considering the memory hierarchy.

From the beginning of this century, the CPU clock frequency is approaching its limit due to the power density limit of the chip. The "free speed up" time has ended. The hardware has shifted to various types of parallelism including vector instructions, multi-core/many-core architectures. Therefore, the performance gain for numerical software application can only be achieved by employing parallelism in software development using code vectorization and multi-threading besides the memory hierarchy optimization.

In this paper, we take the mass amount distribution power flow computation that can be used for Monte Carlo simulation as an example to demonstrate this trend in computing and possible benefit we can obtain by applying performance tuning and parallel programming model.

## III. Distribution System and Power Flow

In this section, we described the basis of the distribution power flow model we used in our computation.

### A. Component Models

In order to preserve the most detail information and achieve the high analysis accuracy, the exact model for three-phase unbalanced distribution system is used in our work. These models are developed by W. Kersting in [3]. To describe the details of the three-phase unbalanced system in phasor domain, the 3 by 1 complex vectors and 3 by 3 complex matrices are used to represent each phase and phase coupling relations. In general, all the components in distribution system can be classified into two catalogues: the two terminal "link models" such as line segments, transformers, and the single terminal "node models", such as spot load. The abstract representations of link model and node model are show in Fig. 2 and Fig. 3.

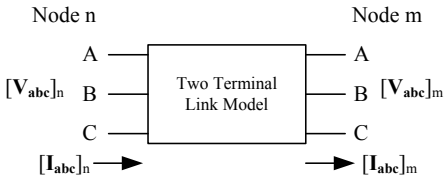


Fig. 2. Link model for two terminal component

For link model in Fig. 2, the relations of the four variables are given as following:

$$\begin{aligned} [\mathbf{V}_{abc}]_n &= [\mathbf{a}]\,[\mathbf{V}_{abc}]_m + [\mathbf{b}]\,[\mathbf{I}_{abc}]_m \\ [\mathbf{I}_{abc}]_n &= [\mathbf{c}]\,[\mathbf{V}_{abc}]_m + [\mathbf{d}]\,[\mathbf{I}_{abc}]_m \end{aligned} \tag{1}$$

Another equation can be also be derived:

$$[\mathbf{V}_{abc}]_m = [\mathbf{A}]\,[\mathbf{V}_{abc}]_n - [\mathbf{B}]\,[\mathbf{I}_{abc}]_m \tag{2}$$

The matrix $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ and $\mathbf{A}, \mathbf{B}$ can be derived from specified equipment models and parameters, these are constant complex matrix in steady state power flow computation.

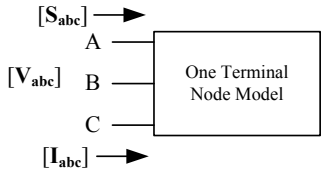


Fig. 3. Node model for one terminal component

For the node model in Fig. 3, considering the three phase complex power $[\mathbf{S}_{abc}]$ injected into the node, the relation between power, voltage and current is:

$$[\mathbf{S}_{abc}] = diag\,[\mathbf{V}_{abc}]\,[\mathbf{I}_{abc}]^* \tag{3}$$

The vector $\mathbf{S}_{abc}, \mathbf{V}_{abc}$ and $\mathbf{I}_{abc}$ represents complex power, voltage phasor and current phasor of each phase.

### B. Solving Power Flow

We only consider the radial structured distribution system. We use the "Ladder Iterative Method" developed in [3] to solve the power flow. The distribution system is represented by a tree structure, the substation is the root of the tree. A Forward Sweep is to update the voltage by traversing the tree from root to leaves, and the Backward Sweep is to update the branch current by traversing the tree from leaves to root. The detail procedure is as following:

Initially, assume the current are all zero, and the voltage are all nominal.

1) **Forward Sweep**: update each downstream node's voltage $\mathbf{V}_{abc}$ using Eqn. 2.
2) **Node Current**: once the new voltages are obtained, using these voltages to compute the current $\mathbf{I}_{abc}$ from each node based on Eqn. 3.
3) **Backward Sweep**: based on the new voltages and currents, using Eqn. 1 to compute each upstream branch's current.
4) **Check Convergence**: Once at the root branch, if the difference of current (or voltage) between two iteration step is within a tolerate threshold, the convergence is reached. Otherwise, go back to Forward Sweep again.

From above computation procedure we can see that the small size complex matrix vector operations are the basic computation kernel. The tree traversal is the basic structure of the computation procedure. These are the main targets of the computation performance tuning.

## IV. Performance Tuning and Parallel Programming Model

In this section, we describe the performance tuning method and the parallel programming model. These are also the two main steps to build our fast distribution power flow solver: the performance tuning part applies code optimization techniques to push the scalar version code's performance to the peak limit using the single core scalar instructions. Based on optimized scalar code, the parallel programming model using Single Instruction Multiple Data (SIMD) instructions and multi-threading can further extract the computation power from hardware by solving many power flow simultaneously.

### A. Performance Tuning

The first step to build an efficient and fast solver is to build an efficient scalar version of the code which fully utilized the hardware computation power in scalar instructions. For the baseline code, we use C++ Standard Template Library (STL), STL provides nice object oriented classes to describe the radial distribution system as a tree data structure. The forward and backward sweep are to iterate over the tree from root to leaves and from leaves to root. The STL is convenient and productive from the software engineering point of view and therefore has been adopted by many software developers. However, these benefits come with the price of performance drop. The extensive point chasing and large amount of overhead often make the typical application's performance one or more orders of magnitude below the processor's capabilities.

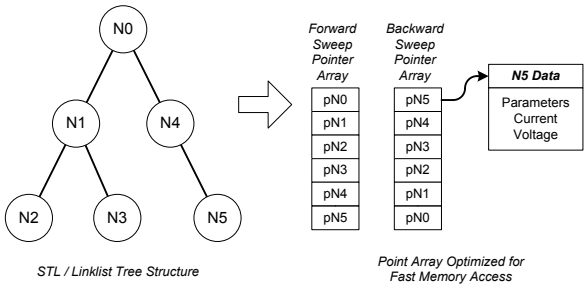


Fig. 4. Shift of data structure for fast memory access

For the data structure optimization, as show in Fig. 4, we convert the STL tree structure to array access and further pointer array access. We put all data into three data arrays: the parameter array, the input array and the output array, and the data needed for forward / backward sweep are placed consecutively in these arrays to exploit the data locality. Further we convert the tree iteration into another two pointer arrays that guide the forward/backward sweep over the data arrays. This optimization strategy converts the pointer chasing to streaming access and preserve the data temporal and spatial locality. Besides data structure conversion, the computation kernel can be further optimized using array scalarization techniques and loop unrolling. The new version scalar code takes advantages of the modern memory hierarchy, and can yield a much better

performance result than baseline STL code. More detailed techniques can be found in [11].

### B. Programming Model

In this application case, we mainly consider the embarrassingly parallelizable Monte Carlo styled applications. The idea is to solve many power flow cases, each case may have different data, but they are all independent from each other. Therefore, mass amount of power flow solvers can be executed simultaneously. Monte Carlo simulation for probabilistic power flow is one of such application case. Besides Monte Carlo simulation, steady state time series simulation, and other statistical applications which require many independent power flow results may also be the application cases.

The parallel programming model of power flow mainly takes advantages of data parallelism and simultaneous multi-threading of modern computing platform. The SIMD instructions enable multiple data being processed by a single instruction simultaneously. On modern mainstream CPUs, the Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) are SIMD instruction set extensions implementation on the x86 architecture. For floating point operations, the SSE enables 4 single precision data being processed simultaneously, and AVX enables 8 single precision data being processed simultaneously, without any extra cost. Besides the SIMD, simultaneous multi-threading enables multiple threads runs on multiply CPU cores simultaneous, with very little overhead cost if the computation workload is high enough. SIMD and multi-threading are two building blocks that enables the utilization of the parallel computation power of modern hardware architecture.

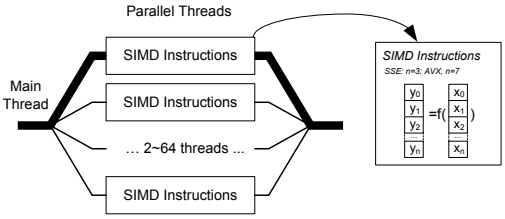


Fig. 5. Parallel programming model on mainstream desktop/server CPU

The overview of the parallel programming model is showed in Fig. 5. We have multiple threads running on multiple CPU cores. Each thread contains one SIMD vectorized power flow solver, depending on the data width of the SIMD instructions, the SIMD vectorized power flow solver can simultaneously solve 4 power flow on SSE instruction set and 8 power flow on AVX instruction set. Multi-threading can further extract the computation power out of the hardware architecture. Multi-threading power flow using Pthread library that enables fast synchronization has been implemented. Ideally, the computation can speedup linearly with the available data width of SIMD instructions and the number of simultaneous CPU threads.

## V. Experiment Result

In this section, we are testing our optimized parallel power flow solver based IEEE 4 bus test feeder system [14]. We expand the system for performance test by duplicating and connecting multiple 4 bus systems together to build larger system up to thousands of buses.

### A. Performance Breakdown

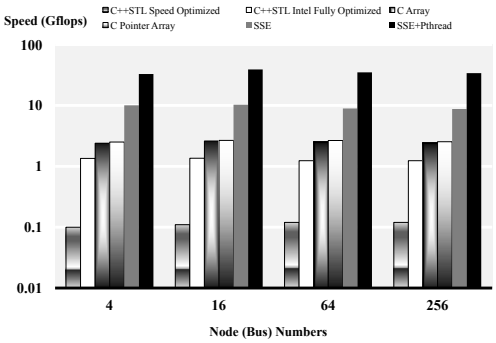


Fig. 6. Impact of performance optimization techniques (on Core2Extreme)

The comparison baseline code is implemented using C++ with various productivity software engineering methods: the operators are overridden for complex number operation, a tree class is developed by using C++ STL template, the tree traverse is an iterator of the template class. An improved C version is implemented by using data array and further pointer array as mentioned in section IV. The Fig. 6 shows the performance breakdown when applying various optimization and parallelization techniques. The lowest bar is the C++ code compiled with compiler's speed optimization options (-o2). The second lowest bar is the same C++ code compiled by Intel Compiler with full compiler optimization options (-o3). The version that uses pointer arrays (the third highest bar) is considered as the best scalar code that can reach nearly 60% of the theoretic peak performance of the CPU scalar instruction hardware. We further implemented the SSE version and SSE with multi-threading version based on this optimized scalar version code, which can give us a nearly linear speedup. From this figure we can see, with only naive software engineering styled implementation and normal compiler options, some two to three orders of magnitude of performance can be lost, which means, more than 99% of the designed computation capability of CPU hardware is wasted for this particular numerical application. With proper optimization and parallelization, we can extract about 50% of the theoretical peak computation capacity of the CPU hardware.

### B. Peak Running Speed

We test our scalar code, SIMD code and multi-thread code on the test feeder system up to the 2048 buses. The speed result on a desktop system with an Intel Core 2 Extreme QX6700 quad-core CPU is showed in Fig. 7. The Composite Theoretical Performance (CTP) of QX6700 is 80.68 Gflops [13]. The optimized Multi-threading SSE code can achieve 50% of

this theoretical peak. Which means on average, the CPU can commit 16 floating point operation per click tick. When the bus number exceed 2048, there is a performance drop due to the cache capacity of the CPU.

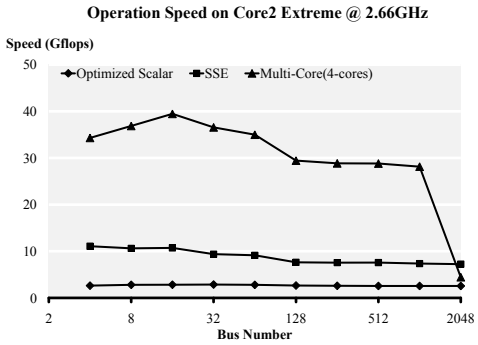


Fig. 7. Performance on one Core2Extreme QX6700 CPU

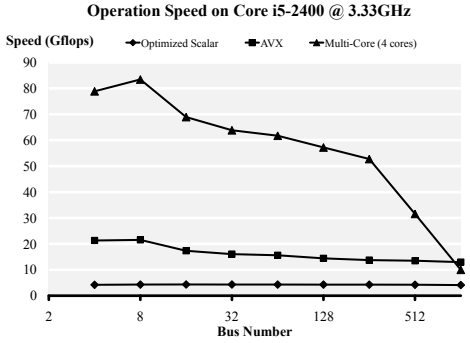


Fig. 8. Performance on Core i5-2400 CPUs with AVX

Fig. 8 shows the performance result on Intel's new Sandy Bridge micro-architecture CPU Core i5-2400, The AVX instruction set is able to process 8-floating point operations per instruction. Comparing to SSE, the AVX can give an almost 2x speedup thanks to the increase of data parallelism. On this CPU, the L2 cache is small than QX6700, therefore, the performance drops at a smaller problem size.

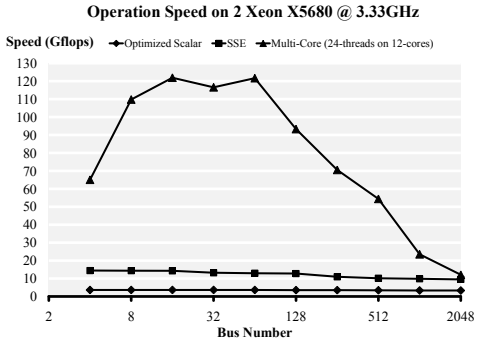


Fig. 9. Performance on two Xeon X5680 CPUs with HyperThreading

Fig. 9 shows the performance result on a server system with two six-core Intel Xeon X5680 CPUs @ 3.33GHz. With HyperThreading, two X5680 enable 24 threads to be executed simultaneously. The parallel programming model implemented in this application enjoys an almost "free speedup" with the increase of the parallel threading capability. This system can delivery up to 120Gflops computation power for specified distribution power flow computation.

## VI. Conclusion

In this paper, we summarized the advances and challenges of modern computing platform for specified power system computation applications. Performance tuning and various forms of parallelization have to be considered for performance oriented applications, without extra work, it is easy to lose several orders of magnitude of performance and waste the CPU's designed computation power. We implemented a high performance parallel distribution power flow solver as an example that shows what performance can be achieved from computation point of view. Our distribution power flow solver is able fully utilize the parallel computing power and can achieve around 50% of theoretical peak performance. Applying parallel programming model to specified power system applications can benefit from the trend of parallelism in computing platform. The "free speedup" in the future may come mostly from the parallel model implemented in specified applications.

## References


[1] T. Chen, M. Chen, K. Hwang, P. Kotas, and E. Chebli, "Distribution system power flow analysis-a rigid approach," *Power Delivery, IEEE Transactions on*, vol. 6, no. 3, pp. 1146–1152, 1991.

[2] C. Cheng and D. Shirmohammadi, "A three-phase power flow method for real-time distribution system analysis," *Power Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 671–679, 1995.

[3] W. Kersting, *Distribution system modeling and analysis*. CRC, 2006.

[4] K. Schneider, D. Chassin, Y. Chen, and J. Fuller, "Distribution power flow for smart grid technologies," in *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*. IEEE, 2009, pp. 1–7.

[5] D. P. Chassin, K. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *Transmission and Distribution Conference and Exposition, 2008. T&D. IEEE/PES*, 2008, pp. 1–5.

[6] B. Das, "Consideration of input parameter uncertainties in load flow solution of three-phase unbalanced radial distribution system," *Power Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 1088–1095, 2006.

[7] Y. Zhu and K. Tomsovic, "Adaptive power flow method for distribution systems with dispersed generation," *Power Delivery, IEEE Transactions on*, vol. 17, no. 3, pp. 822–827, 2002.

[8] N. D. Hatziargyriou, T. S. Karakatsanis, and M. Papadopoulos, "Probabilistic load flow in distribution systems containing dispersed wind power generation," *Power Systems, IEEE Transactions on*, vol. 8, no. 1, pp. 159–165, 1993.

[9] P. Caramia, G. Carpinelli, M. Pagano, and P. Varilone, "Probabilistic three-phase load flow for unbalanced electrical distribution systems with wind farms," *Renewable Power Generation, IET*, vol. 1, no. 2, pp. 115–122, 2007.

[10] E. Gordon, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 4, 1965.

[11] S. Chellappa, F. Franchetti, and M. Püschel, "How to write fast numerical code: A small introduction," *Generative and Transformational Techniques in Software Engineering II*, pp. 196–259, 2008.

[12] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "Top 500 list," http://www.top500.org.

[13] Intel Corporation, "Intel®microprocessor export compliance metrics," http://www.intel.com/support/processors/sb/cs-017346.htm.

[14] W. Kersting, "Radial distribution test feeders," in *Power Engineering Society Winter Meeting, 2001. IEEE*, vol. 2. IEEE, 2001, pp. 908–912.