Performance Studies of a Diakoptics-based Partitioning Approach for Power System Simulation

F. M. Uriarte                      K. L. Butler-Purry

Abstract—Performance studies of partitioned power system simulations on multicore computer are presented. A partitioning approach developed by the authors was used to speed up the simulation of a shipboard power system. The system was torn from its capacitor loops and simulated as subsystems on a quad-core computer. Several performance metrics are presented to assess the performance of this simulation approach: speed gain, accuracy, and core impact. Results from several case studies show the general trend of each performance metric and show that the run-time of large-scale simulations can reduce three orders of magnitude if all cores are used.

Index Terms—diakoptics, model, multicore, partitioning, power, tearing, shipboard, system

I. INTRODUCTION

Several research entities are experiencing simulation run-times of shipboard power system simulations on desktop computers that are unbearable [1-3]. Reasons for such run times are the model order, integration step size, and the frequent network matrix re-factorization required by the presence of many machine and power converter models. What is not always recognized, however, is that partitioning power system models not only divides the work, but it also reduces it.

Efforts to speed up shipboard power system time domain simulations are not new. PSCAD/EMTDC [4] partitions power systems using the travelling-wave method [5], but solves all subsystems on one core. Although the power system is partitioned, its solution is not parallelized. Furthermore, a known limitation of the travelling-wave method is that the simulation time step must be an integer fraction of the travel delay [6]. In shipboard power systems, the short transmission lines would require a time step to \(O(10^{-9})\) or \(O(10^{-12})\), which is impractical and would counteract any speedups achieved from partitioning.

In [7], methods have been thoroughly investigated to partition terrestrial power systems. The MATE formulation makes use of traversal tearing [8],[9], where current injections at the onset of subareas, permit sequentio-parallel solutions. Although MATE has begun to see an application in shipboard power systems [10], the exchange of variables through physical communication networks significantly hinders the potential speed gains.

This work was supported in part by the Office of Naval Research (ONR) grant # N0014-01-1-0760 and a scholarship from the Alfred P. Sloan foundation. The authors are with the Power System Automation Lab of the Electrical and Computer Engineering Department at Texas A&M University, College Station, TX, USA. e-mails: fabian@tamu.edu, klbutler@tamu.edu.
II. THE POWER SYSTEM MODEL

The one-line diagram of the notional AC-radial shipboard power system model taken from [17] is shown in Fig. 1. An enumeration of the power apparatus is given in Table I. The generators are 2.5MW, 450V, 8-pole, 60Hz synchronous machines [18] with delta-connected stators. Each generator is modeled with a prime-mover, governor [19] and an IEEE Type-II voltage regulator and exciter [20]. The induction motors are modeled using approximate per-phase models [21] behind an uncontrolled rectifier and pulse-width modulated three-phase inverter. The cables are modeled as nominal-φ segments without shunt connections to the hull. The loads are modeled as static impedances. The 450:120V transformers are modeled as T-model transformers with Δ-Δ connections. The protective devices are modeled as on/off switches controlled by over-current or under-voltage relays. (For a discussion of protective devices on shipboard power system, the reader is referred to [22].) In addition to the aforementioned power apparatus, nine three-phase faults are distributed across the power system according to the geographical information system presented in [17], which intend to represent abrupt battle damage. Each of the three-phase faults consist are modeled as line-to-line resistances of 50 mΩ each.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEN</td>
<td>Synchronous generator</td>
<td>3</td>
</tr>
<tr>
<td>MOT</td>
<td>Induction motor</td>
<td>19</td>
</tr>
<tr>
<td>RCT</td>
<td>3φ rectifier</td>
<td>19</td>
</tr>
<tr>
<td>INV</td>
<td>3φ inverter</td>
<td>19</td>
</tr>
<tr>
<td>Cbl</td>
<td>1φ cable</td>
<td>33</td>
</tr>
<tr>
<td>CBL</td>
<td>3φ cable</td>
<td>108</td>
</tr>
<tr>
<td>Lod</td>
<td>1φ static load</td>
<td>33</td>
</tr>
<tr>
<td>LOD</td>
<td>3φ static load</td>
<td>13</td>
</tr>
<tr>
<td>XFM</td>
<td>Transformer</td>
<td>11</td>
</tr>
<tr>
<td>BRK</td>
<td>Circuit breaker</td>
<td>83</td>
</tr>
<tr>
<td>ABT</td>
<td>Automatic bus transfer</td>
<td>15</td>
</tr>
<tr>
<td>MBT</td>
<td>Manual bus transfer</td>
<td>13</td>
</tr>
<tr>
<td>LVP</td>
<td>Low-voltage protective device</td>
<td>17</td>
</tr>
<tr>
<td>LVR</td>
<td>LVPs w/recloser</td>
<td>2</td>
</tr>
<tr>
<td>FLT</td>
<td>3φ fault</td>
<td>9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>397</strong></td>
</tr>
</tbody>
</table>
### III. Partitioning Approach

This section briefly introduces the partitioning approach developed by the authors in three stages: discretization, tearing, and implementation. Discretization is presented first because the tearing section assumes a discretized power system model. The tearing section explains how subsystems are formed by tearing capacitors. The implementation section presents network equations at the subsystem level.

#### A. Branch Discretization and Switch Linearization

The power system shown in Fig. 1 was discretized using the EMTP method [23], which replaces all inductors and capacitors with the discrete branches shown in Fig. 2. (Thévenin equivalents are used instead of Norton equivalents to accommodate for a formulation in loop currents as variables.) Referring to Fig. 2, \(\Delta t\) represents the simulation time step, \(k\) the simulation step number, and \(\gamma\) the integration method [24]: \(\gamma = \frac{1}{3}\) for trapezoidal integration; \(\gamma = 1\) for backward Euler integration.

For discretization, the diode with snubber and IGBT with diode and snubber is based on the diode with series capacitance and voltage source

\[
\begin{align*}
\Delta v_i(t) &= L \frac{d}{dt}v_i(t) + \frac{v_i(t) - v_D(t)}{R_s} \\
\Delta v_D(t) &= L \frac{d}{dt}v_D(t) + \frac{v_D(t) - v_i(t)}{R_s} \\
\end{align*}
\]

Discrete equivalent

\[
\begin{align*}
\Delta v_i(t) &= L \frac{d}{dt}v_i(t) + \frac{v_i(t) - v_D(t)}{R_s} \\
\Delta v_D(t) &= L \frac{d}{dt}v_D(t) + \frac{v_D(t) - v_i(t)}{R_s} \\
\end{align*}
\]

Diode w/snubber

\[
\begin{align*}
\Delta v_i(t) &= L \frac{d}{dt}v_i(t) + \frac{v_i(t) - v_D(t)}{R_s} \\
\Delta v_D(t) &= L \frac{d}{dt}v_D(t) + \frac{v_D(t) - v_i(t)}{R_s} \\
\end{align*}
\]

IGBT w/diode and snubber

Fig. 3. Equivalent branches for diodes and IGBTs.

#### B. Capacitor Tearing

The partitioning approach developed by the authors replaces capacitor loops with short circuits to create subsystem decoupling. In the case of a three-phase cable, as shown in Fig. 4, \(C_{ab}\) and \(C_{bc}\) (on either side) are each replaced with a short-circuit. The \{\} symbol on the onset loop currents indicates that more than one loop current can emanate from the left or right. Since all capacitors are discretized \textit{a priori}, the branches that are actually replaced with short circuits are \(R_{Cab2}\) and \(V_{histCab2}\) to represent \(C_{ab}\) and \(R_{Chc2}\) and \(V_{histChc2}\) to represent \(C_{bc}\).

The reason two—and not three—capacitors are torn is because \(R_{Cab2}\) and \(R_{Chc2}\) are the \textit{only two} branches that couple the cable’s internal loop currents to the rest of the network. This realization is particularly effective at buses (i.e., switchboards and load centers) because zeroing-out these two coefficients from the network matrix depletes much of the off-diagonal region redundantly filled with \(R_{Cab2}\) and \(R_{Chc2}\) coefficients. Zeroing-out off-diagonals, as such, produces a block-diagonal structure by tearing a very small number of capacitor loops.

![Three-phase cable model](image)

Fig. 4. Three-phase cable model
This tearing approach is also valid for single-phase cables, and DC links. Consider the DC-link shown in Fig. 6, which could represent the mid-point between a rectifier and an inverter, a DC microgrid bus, or a medium-voltage DC bus as sought for all-electric ships [25]. Discretizing the DC link and replacing $C_{dc}$ with a short-circuit results in the subsystems shown in Fig. 7.

In a power system with hundreds of three-phase cables and many DC buses, where to partition the system should be determined a priori. Two criteria are used to find such disconnection points: the first requires that the resulting $p$ subsystems arising from the partitioning must be—as much as possible—computationally balanced; the second requires that the least number of capacitors be torn.

This type of multi-objective optimization is an NP-complete problem and cannot be expressed in closed form [26]. This tier of the partitioning stage is handled with graph-partitioning software hMetis [27], which finds the best disconnection points from an equivalent graph of a power system. The graph passed to hMetis is a representative weighted graph of the shipboard power system shown in Fig. 1 where each vertex represents one power apparatus, and each edge the junction between power apparatus [28].

C. Network Equations

The loop equations of each subsystem are given in (1)-(2). At each time step of the simulation, the first term on the right-hand side of (1) is solved first. These results are the individual subsystems under the boundary short-circuit condition. These results are used by (2) to compute the voltages across all boundary capacitors as $v^{k+1}_C$. $v^{k+1}_C$ is then impressed onto each subsystem via the second term on the right-hand side of (1). (Readers are referred to [29],[30],[31] for detailed derivation of these diakoptics-based equations.)

$$\begin{align*}
\mathbf{i}^{k+1}_i &= \left( \mathbf{R}^{-1}_C \mathbf{R}_C^{k+1} \right) \mathbf{e}^{k+1}_i + \mathbf{D} \mathbf{v}_p^{k+1} + \text{hist}^{k+1}_{C_i} \\
v^{k+1}_i &= \left( \mathbf{R}^{-1}_C \mathbf{R}_C^{k+1} \right) \mathbf{e}^{k+1}_i + \mathbf{D} \mathbf{v}_p^{k+1} + \text{hist}^{k+1}_{C_i}
\end{align*}$$

\[i^{k+1}_{loop} = \text{loop current vector of subsystem } i\]
\[e^{k+1}_{loop} = \text{voltage impression vector of subsystem } i\]
\[\mathbf{R}_C = \text{diag. matrix of resistances replaced with shorts}\]
\[v^{k+1}_C = \text{vector of voltages across the replaced resistances}\]
\[\mathbf{D} = \ell \times r \text{ transformation tensor:}\]
\[\begin{cases}
1, \text{ if loop current } i \text{ in subsystem } p \text{ enters the } + \text{ side of boundary voltage } j \\
-1, \text{ if loop current } i \text{ in subsystem } p \text{ enters the } - \text{ side of boundary voltage } j \\
0, \text{ if loop current } i \text{ in subsystem } p \text{ does not traverse boundary voltage } j.
\end{cases}\]

IV. MULTITHREADING

The example shipboard power system shown in Fig. 1 was partitioned and simulated on a multicore simulator developed by the author, where each subsystem (1-$p$) was assigned to a different Windows thread. A thread is an independent path of code that can execute concurrently with other threads. The $p$ threads were synchronized using the approach depicted with the swim lane diagram [32] shown in Fig. 8. This swim lane diagram illustrates the work performed by each thread at each time step $k$ of the simulation.

A. Thread Substeps

At time step $k$ of the a simulation, the solver executes eight subroutines (or substeps) $a$ through $h$. In all substeps, except $d$ and $f$, all threads work in parallel and do not require
information from other threads. A description of each substep follows.

![Thread swim-lane diagram](image)

**Substep a:** Each thread updates the coefficient matrix $R_{\text{loop}}$ of its corresponding subsystem. This is necessary if switches, machines, or non-linear elements change state. If this occurs, matrix $A$ in (2) is re-factored during **Substep d**

**Substep b:** The sources in each subsystem are updated. This update corresponds to updating the excitation (i.e., right-hand-side vector $e_{\text{loop}}$) of each subsystem

**Substep c:** Each thread solves the first term on the right-hand side of (1) using an LU decomposition.

**Substep d:** Using solutions 1-2 from **Substep c**, thread 1 computes the boundary condition vector $v_C$ in (2), while threads 2-3 idle. The time spent on this substep limits the maximum speedup arising from partitioned simulations.

**Substep e:** After thread 1 computes $v_C$, each thread patches the result obtained during **Substep c**. This operation corresponds to superimposing the second term on the right-hand side of (1).

**Substep f:** This step checks if power electronic switch current have gone negative, or if their voltages have surpassed their turn-on voltages. If so, the solution obtained at **Substep e** is time-interpolated back to the first switch transition before continuing to **Substep f**. (Details on interpolation techniques can be found in [33].)

**Substep g:** Using the solution from **Substep e** (or **Substep f** if a switch commutated), the voltages and currents in each subsystem are computed.

**Substep h:** The measurements made in **Substep g** for all protective devices and user-specified measurements are recorded.

### B. Thread Distribution

The $p$ threads can be assigned to the cores of a multicore processor in two ways: manually by using each thread’s affinity property, or automatically by indicating that Windows should distribute the threads as it deems best. To assess the impact of incrementing the number of cores utilized, a manual distribution was selected.

Before running a simulation, the number of partitions must be chosen automatically. To do so, the program chooses the number of partitions using the criteria specified in (3), which for the system and formulation studied, consistently resulted in the best performance. In (3), $r$ corresponds to the order of matrix $A$ in (2), and $O(R_{\text{loop}})$ represents the order of the $i$th subsystem’s loop resistance matrix. Equation (3) states that the best performances occurs when the total number of capacitors replaced with short circuits nears 1/3 of the average subsystem order. This is an empirical result valid for the system and formulation under study, and does not intend to be general.

$$r_{\text{avg}} = \frac{r}{\sum_{i=1}^{p} O(R_{\text{loop}})} \approx 33\% \quad (3)$$

### V. PERFORMANCE METRICS AND RESULTS

This section presents performance metrics to assess several aspects of the partitioning approach of the authors. The results presented are general trends, which are generalized, typical results observed from several case studies. The case studies differed in component count, system topology, and power apparatus complexity. Varying a system model as such permits estimating the results for a wide range of system sizes and complexities instead of presenting a case-specific result.

In all case studies the power system model in Fig. 1 was simulated from zero initial conditions until the steady state was reached. When in steady state, the faults were applied at the locations indicated in Fig. 1. In some cases, the faults were applied simultaneously; in other cases, sequentially. It is noted that the partitioning method and performance metrics presented here are independent of the power system model. A shipboard power system is used to illustrate the results, but the work is not limited to architectures of this type.

The simulation end time and time step increment in all cases were set to $t_{\text{end}} = 1$ s and $\Delta t = 50$ µs, respectively. This combination of time parameters results in $1/50 \times 10^6 = 20 \times 10^3$ simulation time steps, which suffices to generalize the results. Each performance metric and their results follow.

#### A. Speed Gain

The simulation speed gain is a standard metric in parallel computing [34], and is used to determine whether partitioning a problem reduces its run-time or not. The speed gain $K_{\text{speed}}$ was computed as the run-time ratio in (4), where $t_{\text{unpartitioned}}$ and $t_{\text{partitioned}}$ are the unpartitioned and partitioned run-times in seconds, respectively.

$$K_{\text{speed}} = \frac{t_{\text{unpartitioned}}}{t_{\text{partitioned}}} \quad (4)$$
The general pattern of the speed gain results are shown in Fig. 9 for $c=1$ through $c=4$ cores (a quad-core machine was used). These results are averaged values obtained over several case studies under different system sizes, configurations, and complexity.

Referring to the cases where one core ($c=1$) was used, speed gains of $\geq O(10^3)$ are possible starting from $p=2$. This result indicates that partitioning alone, without going multicore (i.e., $c=1$), suffices to reduce simulation run-time—but more is possible if all cores are used (e.g., $p=4$, $c=4$). It was also observed that even with one core, the speed gains are super-linear (i.e., $K_{\text{speed}} > p$ for $2 < p < 16$). Diminishing returns are noticed when $p>8$ for $c=1$. Super-linear gains are possible because the problem size reduced as $p$ increased—independently how many cores were used.

![Fig. 9. Average speed gains for all number of partitions and cores](image)

In the $c=2$, $p=2$ case, the speed gain of $K_{\text{speed}} = 29$ also reached $O(10)$. This result is more than twice the $c=1$ case. The speed gains for the $c=2$ case continued increasing through $p \approx 10$, where diminishing returns were first observed. The $c=3$ case, recorded as of $p>3$ reached two orders of magnitude at $p=4$ (i.e., $K_{\text{speed}} = 111$). Similar to the $c=2$ case, the speed gain for $c=3$ increased steadily through $p=8$, and then started to diminish—but at lesser rate than the $c=1$ and $c=2$ cases.

The $c=4$ case, which starts from $p=4$, does not offer much more speed gain when compared to the $c=3$ case—that is, there is not much of a relative gain between $c=3$ and $c=4$. This result arises when manual thread distributions are enforced, which hinders the speed gain potentials of multicore computers. (A contrast of the core usage between a manual and automatic thread distribution is shown later in Fig. 12-Fig. 14.) In all number of partitions, however, maximizing $c$ maximized the simulation speed.

### B. Accuracy

The accuracy of partitioned simulations was assessed by computing the %-error between partitioned and unpartitioned simulation results with (5). In (5), $x_1^{k,\text{unpartitioned}}$ represents a data point (i.e., voltage or current) from an unpartitioned ($p=1$) simulation run, and $x_1^{k,\text{partitioned}}$ the corresponding data point from a partitioned simulation (i.e., $2 < p < 12$). (The number of cores is not relevant to this performance metric.)

$$
e = \frac{100}{x_1^{k,\text{unpartitioned}}} \left| x_1^{k,\text{partitioned}} - x_1^{k,\text{unpartitioned}} \right|$$  \hspace{1cm} (5)

The minimum, average, and maximum simulation errors for all $p$ were averaged for several case studies. The results are presented in Fig. 10. The errors observed are negligible, and of fractional percent. A way to assess whether the errors are influential in partitioned simulations is by the RMS measurement of each protective device. Incorrect RMS measurements due to bad data cause loss of security in protective devices (i.e., false tripping). The maximum error found, which was $O(10^{-1})\%$, did not cause bad data.

![Fig. 10. Average error for all number of partitions](image)

The maximum error is a peak mismatch that occurred once. Of more importance than the peak error is the average error, which represents the likely mismatch when two arbitrary data points are compared. As noted from Fig. 10, the average error was also negligible, and did not affect the results. When waveform traces are overlaid, all curves appear visually lined-up over one another, which is another shows that warrants accuracy from a pragmatic perspective. However, computing the error arithmetically can identify errors in places not plotted.

### C. Core Impact

The core impact measures how the simulation run-time improves as the number of cores used increases by one. (A manual thread distribution is required to assess this performance metric instead of allowing Windows to do so.) The core impact when using two cores ($c=2$) is the relative speed gain $K_{\text{speed}2} / K_{\text{speed}1}$, where $K_{\text{speed}}$ is the speed gain when $c=2$, and $K_{\text{speed}1}$ is the speed gain when $c=1$. The core impact with any number of cores is given by (7), where $K_{c,1}$ is the core impact, and $K_{\text{speed}c}$ is the speed gain when $c$ cores are used.

$$K_{c,1} = K_{\text{speed}c} / K_{\text{speed}1}$$  \hspace{1cm} (6)

The maximum speed gains per core and relative speed gains with respect to $c=1$ are shown in Fig. 11. When $c=2$, $K_{2,1} = 3.05 > c$, which confirms the super-linear gains explained for Fig. 9. This results indicates that the simulations execute approximately three times faster when $c=2$ are used.
over \( c=1 \). Although when \( c=1 \) the speed gains were 56x, incrementing from \( c=1 \) to \( c=2 \) three-folds this number (i.e., max. speed gain with \( c=2 \) was \( K_{\text{speed}} = 56 \times 3.05 = 171 \)).

When increasing from \( c=1 \) to \( c=3 \), the maximum speed gain increased observed with \( c=1 \) increased twenty times \( K_{\text{speed}} = 56 \times 19.08 = 1109 \), which is a significant difference.

The impact of going from \( c=1 \) to \( c=4 \) is significant, but not from \( c=3 \) to \( c=4 \). This result suggests that a manual distribution of threads is not an good way to distribute the work on a multicore processor; however, it does permit assessing the core impact.

While it is difficult to determine the main contributor of the core impact, there are several factors that influence this result. These factors include type of processor used, number of threads assigned to the simulator’s process, the priority level of each thread, computational imbalance, thread synchronization times [35], programming efficiency, and other Windows background processes demanding processor time.

D. Core Usage

The core usage shows how well a multicore processor is utilized. Fig. 12 shows an example core usage for when \( c=1 \) and \( p=8 \). The total processor usage was 27%, which emanates mainly from core 1. Cores 2-4 are not working towards the simulation results, which results in processor underutilization.

When the number of cores used increased from \( c=1 \) to \( c=4 \), 86% of the processor was used as shown in Fig. 13. Although all cores were used, the processor was fully utilized. The best processor occurs when Windows is allowed to distribute the threads onto each core. For example, consider the simulation case shown in Fig. 14. In this case, all cores are fully utilized and performance maximized. Although this type performance is normally desired, the thread distribution is unknown—that is, the way Windows mapped the threads to each physical core is unknown.

VI. SUMMARY AND CONCLUSIONS

A power system partitioning approach was presented and assed on a multicore computer using several performance metrics. The partitioning approach replaces capacitors with short circuits to produce subsystems by short-circuit decoupling. This approach is particularly effective at buses where many partitions can be created from the same boundary keeping the number of boundary variables small.

The performance metrics assessed were the speed gain, accuracy, core usage, and core impact. The speed gain results showed super-linear gains in several cases. This result suggests that it is better to partition a power system model and execute it on a several cores, rather than executing parallelizing unpartitioned cases on each core. The major bottleneck of the partitioning algorithm corresponds to substeps \( a, d \) and \( f \). These substeps correspond to re-factoring the network matrix, computing the boundary variables, and interpolating the solution. These three substeps become increasingly pronounced as the number of power electronic switches increases. The relative core gain showed the incremental gains when increasing the number of cores by one. Although there are significant gain differences when increasing the number of cores \( c=1 \) to \( c=4 \), the best performance is obtained when Windows is allowed to distributed the threads automatically.

It is concluded that partitioning power systems and using multicore computers to perform intensive time domain simulations is a cost-effective approach that can postpone (and even circumvent) the acquisition of costly simulation hardware. The potential of multicore computers is normally overlooked, but is growing in importance. It is common that users manifest marginal speed gains despite multicore computer upgrades. As this work has shown, and exemplified via a power system model of sufficient complexity and order, power system simulations run-times can be significantly reduced if the solution is partitioned, parallelized, and properly distributed to the cores of a multicore computer via a multithreaded solver.

REFERENCES


