Using the Cell Broadband Engine to Compute an Electrochemical Battery Model

James Geraci[1] Sudarshan Raghunathan[1] John Chu[1]

[1] Massachusetts Institute of Technology

Abstract

This paper discusses the usage of a two-dimensional electrochemical battery model of a lead acid battery cell as a state-of-health indicator in an embedded application. The model equations are a set of coupled highly non-linear partial differential equations that evolve with time. At each time step, these equations are solved using Newton's Method, and a direct skyline LU solver without partial pivoting is used to solve for the battery cell's state at each Newton step.

The entire model was implemented on a Sony Playstation 3 and the direct solver at each Newton iteration step exploits the parallelism in the Cell Broadband Engine to improve performance. The parallelized direct solver outperforms state-of-the-art dense and sparse solvers running on an AMD Opteron 246-based workstation.

1 Introduction

With the increased interest in hybrid and electric cars, there is also an increased interest in answering not only the question of what is the state-of-charge of a battery but also, what is the state-of-health of the battery? While there are numerous empirical models that allow estimates of the state-of-charge of a battery [1, 2, 3, 4, 5, 6], it is much more difficult to determine the state-of-health of a battery[7].

The state-of-health of a battery gives some idea of the how different the battery at the present time is from the battery when it was new. This difference between the battery at a time $t = T$ and $t = 0$ depends greatly on how the battery is used. For example, a typical flooded lead acid battery has about 1500 uses when used to a depth of discharge of 70% while a typical VRLA (Valve-Regulated Lead Acid) battery only has about 400 to 500 cycles at the same depth of discharge[8]. However, to about 2200 uses and 1000 uses respectively if they are only discharged to 40% depth of discharge [8]. The number of uses (cycles) that a battery can go through is called the cycle life of the battery.

Not only is cycle life a strong function of how the battery is used, but also "cycle life is dependent on a number of construction factors"[9] These construction factors include, but are not limited to: the thickness of the plates, the active mass density of the active material, the concentration of the acid, and the geometry of the electrodes[9, 10]. Therefore, it would be helpful to have a model that can take these parameters into account. In the case of the lead acid battery, a two-dimensional electrochemical model such as that found in [11], and modified in [12] to include a state-of-health metric, would take all of these parameters into account.

We are not aware of any previous implementations of a two-dimensional electrochemical battery model that have been used in an embedded application to help provide information about the state-of-health of a battery. This is in part due to the notion that these models require too much computational effort to be cost effective in an embedded environment. However, with the new multicore processors that the computer industry is racing to introduce, the cost and power consumption required to achieve multi-gigaflop performance is rapidly decreasing. For example, the new Intel Polaris processor is a 275 square mm collection of 80 extremely simple cores that are collectively capable of 1 teraflop while consuming only 62W of power. As a reference, the first teraflop computer ASCI Red came into service in 1997 at Sandia National Laboratories "It was 104 cabinets housing 10,000 Pentium Pros and spread out over 2500 square feet. It consumed a mere 500kW"[13].

The Cell Broadband Engine used in the Playstation 3 represents the first truly multicore processor based on the simple core approach available to the public. This processor is capable of about 200 Gflops single precision and around 20 Gflops double precision performance.

1.1 Outline

This article first takes a look at the electrochemical model used and the types of output one can get from the model. Then it looks at the depth of discharge based state-of-health indicator. Next, the numerical algorithm used for simulating the model is described along with a description of the implementation on the Cell Broadband Engine. The performance and scaling results are then reported and com-

pared with existing dense and sparse solvers on a contemporary desktop workstation. Finally, conclusions are presented along with venues for further investigation.

2 Two-Dimensional Battery Model

The electrochemical battery model used in this study is based on the model given in [11] and derived from first principles in [12].

A typical lead acid battery, consisting of a lead dioxide electrode, a lead electrode, and a liquid sulfuric acid electrolyte is illustrated in Figure 1. The model tries to simulate the primary electrochemical reactions that occur during charge and discharge in the area of an electrochemical cell between the lead and lead dioxide plates (indicated by the dotted region in Figure 1).

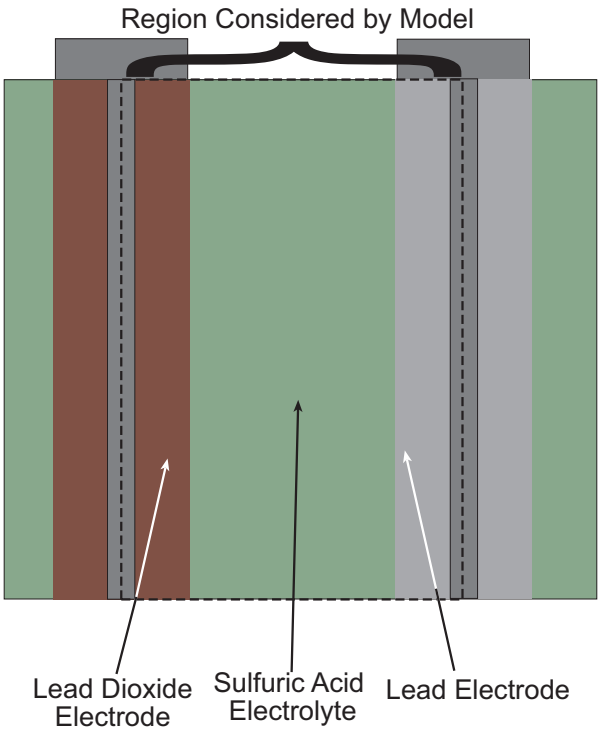

Figure 1. A complete lead acid battery cell.

The model has four state variables that evolve with time: the porosity of a region ε, the concentration of the electrolyte c, the liquid phase electrical potential ϕ_l, and the solid phase electrical potential ϕ_s.

The spatial region of interest is discretized into two different staggered two-dimensional grids of volumes as seen in Figure 2. This technique is known as a *staggered grid*. Three of the state variables c, ϕ_l, and ϕ_s were centered on one grid (called the PV grid for potential values), while ε was centered on the other grid (called the FV grid for flux values). In contrast to the model described in [11], the staggered grid approach for the potential and flux values avoids having to enforce continuity conditions at the electrode/electrolyte boundaries.

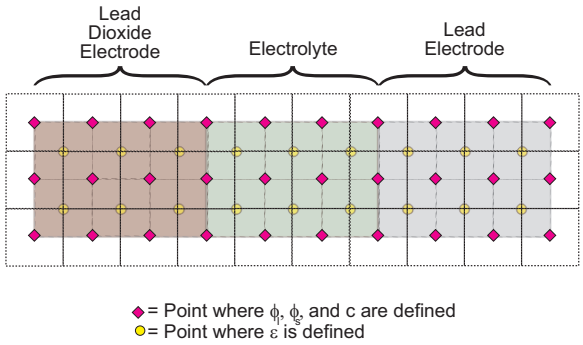

Figure 2. A battery cell with both the PV grid and FV grids shown. Diamonds show the center of the PV grid while circles show the center of the FV grid.

The variables on the PV grid are solved for at each time step as a coupled set of non-linear equations using Newton's method. However, the porosity ε is not included in the Jacobian for each Newton step as it is on the FV grid.

Finally, for each PV volume, there is a depth-of-discharge (DOD) variable that counts the number of times the concentration in that volume has dropped below a predetermined level.

2.1 Output

The state variable values at each time step are saved for post-processing. Figure 3 shows how the concentration of electrolyte, c, might look throughout the cell after 800 seconds of discharge at a rate of 1^{-3}Amps. Figure 4 shows the same cell as if it had been discharged at a rate of 1Amp for 800 seconds. Here it can been seen that some of the volumes of the upper portion of the lead electrode have started to fail.

The failed volumes in Figure 4 are actually volumes that have had their solid-liquid interface area set to zero by the state-of-health (SOH) indicator. For demonstration purposes, the DOD counter has been set to count the amount of time spent under a certain DOD threshold instead of the number of times that particular DOD threshold has been crossed. When a particular volume has spent more than a predetermined amount of time under a certain depth of discharge, the SOH indicator sets the solid-liquid interface area for that volume to zero. This stops the kinetics in that particular volume but allows electronic and ionic current to continue to pass through the volume.

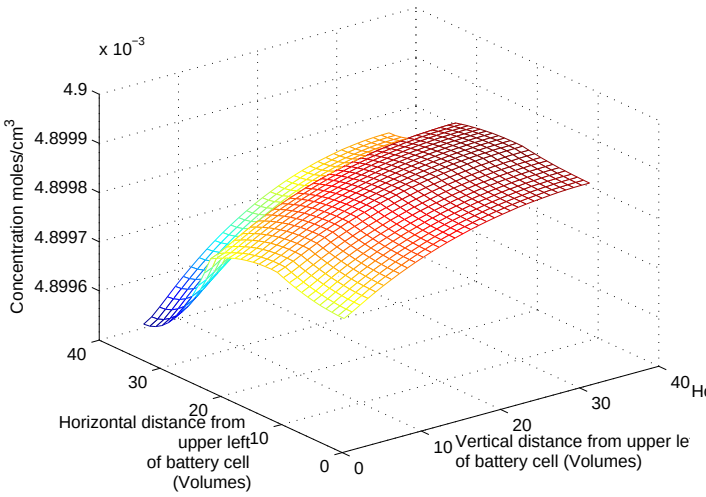

Figure 3. Concentration of electrolyte c within each PV of the battery cell after the battery cell has been discharged at a rate of 1^{-3} Amps for 800 seconds. The point located at coordinates $(0, 0)$ corresponds to the upper left corner of the region described by the model as seen in Figure 1

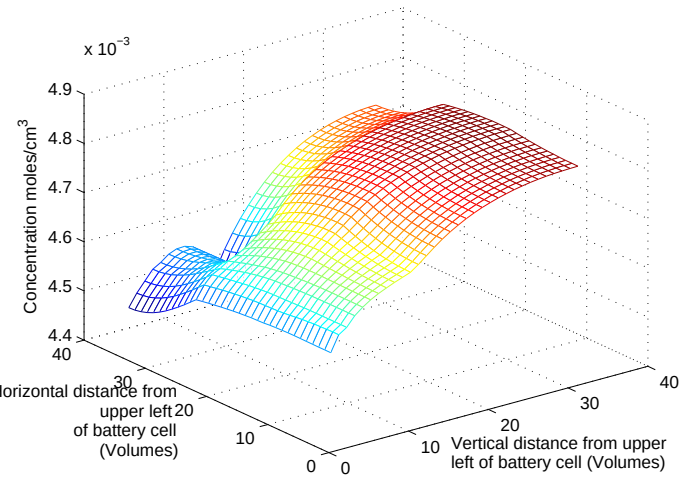

Figure 4. Concentration of electrolyte c within each PV of the battery cell after the battery cell has been discharged at a rate of 1 Amp for 800 seconds. It can be observed that some of the finite volumes have started to fail.

An SOH metric for the entire battery cell can be implemented based on all the SOH status of the individual volumes. For example, when 20% of the cells have an SOH status of off, the battery might be said to have failed.

3 Algorithm and Implementation

Implicit time stepping was used for the variables on the PV grid, and the values of the state of the variables on the PV grid were solved for at each time step using Newton iteration. Each Newton iteration requires the solution of a system of the form $Jx = r$, where J is the Jacobian matrix produced by the system, see Figure 5, and r is the residual vector.

The Jacobian for this system has a condition number of approximately 10^8. This is in part due to the weak coupling between the volumes in the y-direction due to the large height of the battery cell (10.0 cm) when compared to the small width of the battery cell (0.09 cm). Due to the large spacing in the y direction, the reference potential, chosen to be ϕ_l in the lower right corner of the lead electrode, has only a weak influence on the values at other points in the system in the y-direction.

Since the Jacobian has a condition number of approximately 10^8, there are effectively only 8 digits of useful precision in any answer, consequently, the model is said to converge when the largest r value is less than 10^{-8}. Furthermore, such a poor condition number does not often work well with an iterative solver, so a direct solver is used to compute x at each Newton iteration.

The direct solver has two major computational phases: forward elimination followed by back substitution. The forward elimination part is more computationally intensive, $\mathcal{O}(b^2N)$ (where N is the size of the Jacobian and b is the half-bandwidth) and is therefore performed in parallel on the Synergistic Processing Units (SPUs) of the cell processor with the Power Processing Unit (PPU) being used for synchronization. The back substitution phase is computationally less expensive ($\mathcal{O}(bN)$) and is therefore done completely on the PPU.

Pseudocode for the forward elimination step running on the SPUs is given in Algorithm 1. For conciseness, Algorithm 1 only shows the operations on the Jacobian; a similar set of operations is used to update the corresponding component of the residual.

In the actual implementation, the Jacobian is assembled in the main memory of the PS3 as a dense matrix with elements stored in row-major order. This scheme ensures unit stride access to the non-zero elements in each row by the SPUs during elimination. The actual code that runs on the SPUs is double buffered and employs SIMD instructions to maximize computation efficiency.

Input: The Jacobian matrix, $\mathbf{J}$ at a Newton iteration with half-bandwidth b and the residual vector, $\mathbf{r}$
Result: Forward elimination is performed on the Jacobian and the residual vector

```
for i ⟵ 1 to N − 1 do
    // Fetch base row i from main memory
    J_local,i = post_recv(J[P[i], i : i + b])
    // Fetch first elimination row for this SPU, i + spuid from main memory
    if i + spuid ≤ N then
        J_local,i+spuid = post_recv(J[P[i + spuid], i + spuid : i + spuid + b])
    end
    // Wait for base row and first elimination row to arrive
    wait_for_completion(J_local,i, J_local,i+spuid)
    for j ⟵ i + spuid to i + b do
        // Pre-fetch next elimination row for this SPU, j + spuid from main memory
        if j + spuid ≤ N then
            J_local,j+spuid = post_recv(J[P[j + spuid], j + spuid : j + spuid + b])
        end
        // Perform elimination on row j
        J_local,j[i] ⟵ J_local,j[i]/J_local,i[i]
        for k ⟵ i + 1 to i + b do
            J_local,j[k] ⟵ J_local,j[j] × J_local,i[k]
        end
        // Post the updated row back to main memory
        post_send(J_local,j)
        // Wait for all pending posts
        wait_for_completion(J_local,j)
        // Wait for next elimination row to arrive
        if j + spuid ≤ N then
            wait_for_completion(J_local,j+spuid)
        end
    end
    // Wait before starting next base row
    wait_for_notification
end
```

Algorithm 1: Algorithm for parallel forward elmination step of the banded direct solver.

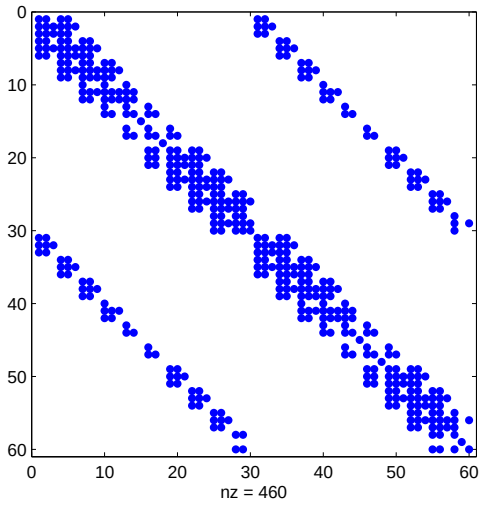

Figure 5. Location of the non-zero entries in the Jacobian. In this case, the Jacobian is a 60 $\times$ 60 with matrix with 460 non-zero entries. The skyline form of the matrix can be clearly seen.

4 Performance results

Figure 6 illustrates the performance of the direct banded solver for Jacobians of different sizes with increasing number of SPUs. It is observed that while the algorithm does scale with increasing number of SPUs, the speedup is sub-linear beyond two SPUs. This can be attributed to the increase in time it takes the PPU to synchronize the increasing numbers of SPUs as seen in Figure 7.

Figure 8 shows the performance of the parallel direct solver relative to existing sparse and dense solvers in UMFPACK [14] and LAPACK [15] respectively for a Jacobian matrix of size 3264×3264.

Even though the technique used in each of the solvers is different, for consistency the performance in GFlops is measured using the operation count for solving a dense linear linear system of order N:

$$\text{Gflops} = \frac{\frac{2}{3}N^3 - \frac{1}{2}N^2}{\Delta t \times 1024^3} \tag{1}$$

where Δt is the total amount of time to perform forward elimination and back substitution.

The results for UMFPACK and the PS3 based solver can be interpreted as being the performance of a dense solver that would be needed in order to achieve the Δt achieved by UMFPACK or the PS3 based solver.

Figure 8 shows that the PS3 based solver is 1.56x faster than UMFPACK when 2 SPUs are used. However, due

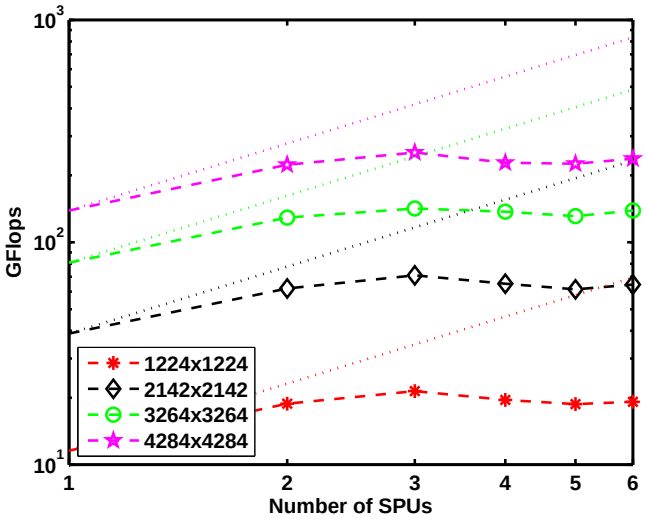

Figure 6. Performance in Gflops achieved by the CBE based LU algorithm. To maintain consistency between this figure and Figure 8 Gflops has been computed using (1), the equation for a dense solver even though the PS3 solver is a banded solver.

to the extra synchronization overhead incurred when using more SPUs, there is no performance to be gained when using more than 2 SPUs.

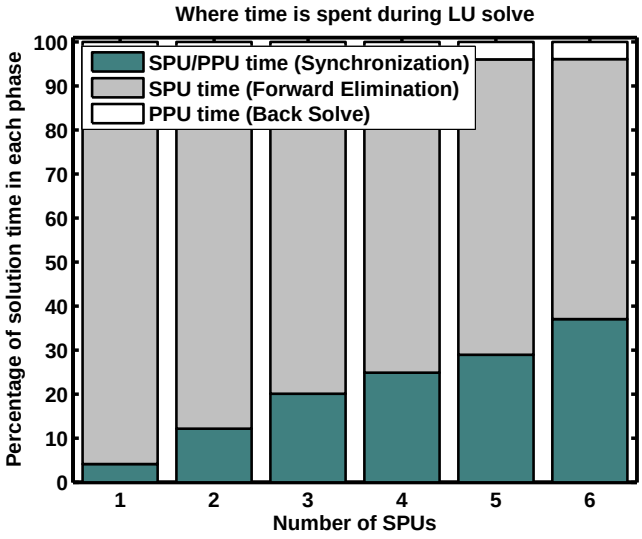

Figure 7. Three major time consuming steps for the sparse LU solver on the PS3. These are forward elimination, back solve and SPU synchronization. As the number SPUs is increased, the percent of time spent synchronizing the SPUs increases dramatically.

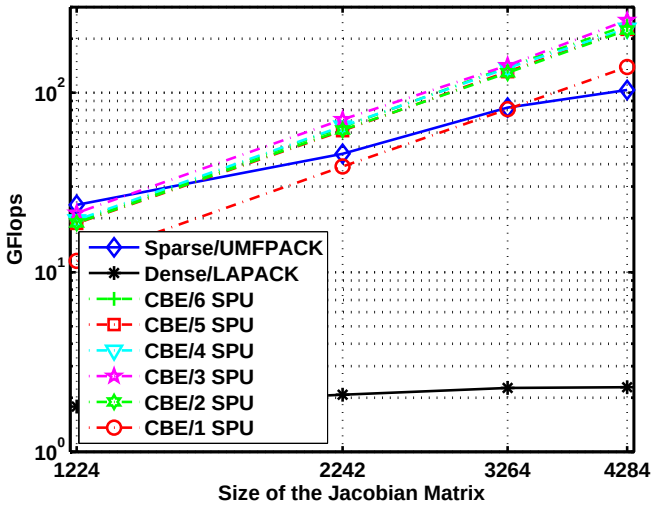

Figure 8. A comparison of the performance of LAPACK, UMFPACK, and the PS3 based solver. All performance numbers were computed using (1).

5 Conclusions and further work

In this article, an numerical algorithm for the simulation of a two-dimensional electrochemical model of a lead acid battery derived in [12] was presented. The most computationally expensive component of this model is the solution of a banded linear system of equations during each Newton iteration step. Therefore, an algorithm for solving this system in parallel on a Cell Broadband Engine was presented. The proposed approach achieved performance competitive to existing sparse solvers running on modern desktop workstations. This suggests that the Cell Broadband Engine can be effectively used in low-power embedded applications requiring significant computational resources.

To further improve the proposed algorithm and to reach the true performance of the cell processor, it is critical that the synchronization mechanism between the Synergistic Processing Units be improved. Finally, it is hypothesized that fetching rows into the SPUs using triple instead of double buffering will lead to further performance increases.

References

[1] S. Piller, M. Perrin, and A. Jossen. Methods for state-of-charge determination and their applications. *Journal of Power Sources*, 96:113–120, 2001.

[2] B. Le Pioufle, J.F. Fauvarque, and P. Delalande. A performing lead acid cell state of charge indicator based on data fusion. *Electrochemical Society Proceeding*, 16, 1996.

[3] V.H. Johnson. Battery preformance models in advisor. *Journal of Power Sources*, 4806:1–9, 2002.

[4] Wootaik Lee, Daeho Choi, and Myoungho Sunwoo. Modelling and simulation of wehicle electric power system. *Journal of Power Sources*, 109:58–66, 2002.

[5] A.H. Anbuky and P.E. Pascoe. VRLA battery state-of-charge estimation in telecommunication power systems. *IEEE Transactions on Industrial Electronics*, 47(3):565–573, 2000.

[6] S. Rodrigues, N. Munichandraiah, and A.K. Shukla. A review of state-of-charge indication of batteries by means of a.c. impedance measurements. *Journal of Power Sources*, 87:12–20, 2000.

[7] A. Delaille, M. Perrin, F. Huet, and L. Hernout. Study of the ”coup de fouct” of lead-acid cells as a function of their state-of-charge and state-of-health. *Journal of Power Sources*, 2006.

[8] K. Peters. Review of factors that affect the deep cycling performance of valve-regulated lead/acid batteries. *Journal of Power Sources*, (59):9–13, 1996.

[9] P.Ruetschi. Aging mechanisms and service life of lead-acid batteries. *Journal of Power Sources*, (127):33–44, 2004.

[10] D. Berndt. Valve-regulated lead-acid batteries. *Journal of Power Sources*, (100):29–46, 2001.

[11] D. M. Bernardi and H. Gu. Two-dimensional mathematical model of a lead-acid cell. *Journal of Electrochemical Society*, 140(8):2250–2258, 1993.

[12] J. Geraci. *Electrochemical Battery Models*. PhD thesis, Massachusetts Institute of Technology, 2007.

[13] C.Demerjian. Intel 80 core chip revealed in full detail. http://www.theinquirer.net/default.aspx?article=37572, February 11 2007.

[14] T.A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.

[15] *LAPACK UsersǴuide*. SIAM, 1999.