High Performance Simulations of Electrochemical Models on the Cell Broadband Engine

James Geraci[1] Sudarshan Raghunathan[2]
[1] Massachusetts Institute of Technology
[2] Interactive Supercomputing

Abstract

This article describes the use of the cell broadband engine as a high-performance platform for simulating the discharge of a lead acid battery to determine its state of health in an embedded application.

The discharging of the battery is modelled as a two-dimensional electrochemical process. 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 linear solvers running on desktop workstations.

1 Introduction

With the increased adoption of hybrid and electric cars, there is also an increased interest in determining in real time, important parameters about automobile batteries such as the state of charge, state of health and remaining cycle life. Unfortunately, an accurate assessment of these parameters requires computationally expensive mathematical models such as those described in [1]. Due to the power-hungry nature of existing high-performance workstations, the use of accurate models of automobile batteries in embedded applications was until recently not possible. However, the emergence of low-power multi-core chips such as the Cell Broadband Engine has put multi-gigaflop performance within reach of several embedded applications.

In this article we describe the parallel implementation of a two-dimensional electrochemical model of a lead acid battery on the Cell Broadband Engine. We first give a brief overview of the model, followed by the description of the parallel sparse direct solution algorithm and its implementation on the Cell processor. We the compare its performance and scaling characteristics with respect to state of the art dense and sparse solvers on existing high-performance workstations. Finally, we present our conclusions along with avenues for further investigation.

2 Model and Parallel Implementation

The mathematical model used for simulating the discharge of a lead acid battery is described in more detail in [1, 2] and references therein.

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 a staggered finite volume grid, with the electrolyte concentration and the liquid and solid phase electrical potentials being represented on the PV (potential value) grid and the porosity on the FV (flux value) 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 and the corresponding values in the FV grid are then obtained directly as described in [1]. Each Newton iteration requires the solution of a system of the form $Jx = r$, where J is the Jacobian matrix and r is the residual vector. For the particular model under investigation and the chosen discretization scheme, the Jacobian is banded and has a condition number of roughly 10^8. It is therefore most effeciently solved using a sparse direct solver.

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 on the Jacobian is given in Algorithm 1. The actual code that runs on the SPUs is double buffered and employs SIMD instructions to maximize computation efficiency. Note that due to the poor conditioning of the Jacobian, all computations are performed directly in double precision.

3 Performance results

The performance characteristics of our sparse direct solver are illustrated in Figure 1 (a) and (b).

It is observed that while our algorithm does scale with increasing number of SPUs, the speedup is sub-linear beyond two SPUs. This can be attributable to the increase in time it takes the PPU to synchronize the increasing numbers of SPUs. However, even with the increased synchronization overhead, it is observed in Figure 1 (b) that the parallelized sparse solver on the cell processor is around 50% faster than UMFPACK [3] and several hundred times faster than LAPACK for a linear system of size 3264×3264.

4 Conclusions

In this article, an numerical algorithm for the simulation of a two-dimensional electrochemical model of a lead acid battery derived in [1] 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 clearly outperforms 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.

References

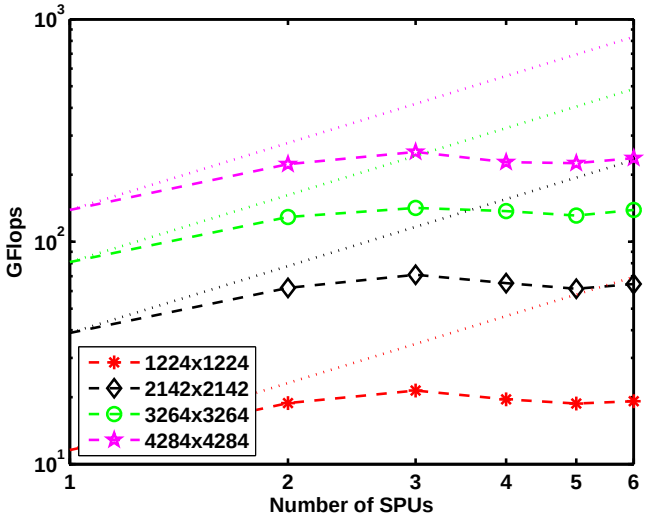
[1] J. Geraci. *Electrochemical Battery Models*. PhD thesis, Massachusetts Institute of Technology, 2007.

[2] J. Geraci, R. Sudarshan, and J. Chu. Using the cell broadband engine to compute an electrochemical battery model. Technical report, Research Lab for Electronics, Massachusetts Institute of Technology, 2007.

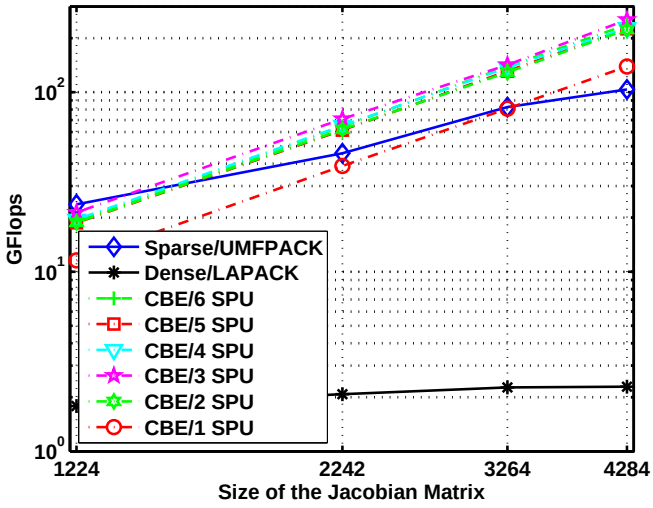
[3] T.A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.

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 $i \longleftarrow 1$ ***to*** $N-1$ **do**
  //Fetch base row i from main memory
  $\mathbf{J}_{\text{local},i} = \texttt{post\_recv}(\mathbf{J}[\mathbf{P}[i], i : i+b])$
  //Fetch first elimination row for this SPU, $i + \texttt{spuid}$ from main memory
  if $i + \texttt{spuid} \leq N$ **then**
    $\mathbf{J}_{\text{local},i+\texttt{spuid}} = \texttt{post\_recv}(\mathbf{J}[\mathbf{P}[i+\texttt{spuid}], i+\texttt{spuid} : i+\texttt{spuid}+b])$
  end
  //Wait for base row and first elimination row to arrive
  $\texttt{wait\_for\_completion}(\mathbf{J}_{local,i}, \mathbf{J}_{local,i+\texttt{spuid}})$
  for $j \longleftarrow i + \texttt{spuid}$ ***to*** $i+b$ **do**
    //Pre-fetch next elimination row for this SPU, $j + \texttt{spuid}$ from main memory
    if $j + \texttt{spuid} \leq N$ **then**
      $\mathbf{J}_{\text{local},j+\texttt{spuid}} = \texttt{post\_recv}(\mathbf{J}[\mathbf{P}[j+\texttt{spuid}], j+\texttt{spuid} : j+\texttt{spuid}+b])$
    end
    //Perform elimination on row j
    $\mathbf{J}_{\text{local},j}[i] \longleftarrow \mathbf{J}_{\text{local},j}[i] / \mathbf{J}_{\text{local},i}[i]$
    for $k \longleftarrow i+1$ ***to*** $i+b$ **do**
      $\mathbf{J}_{\text{local},j}[k] \longleftarrow \mathbf{J}_{\text{local},j}[j] \times \mathbf{J}_{\text{local},i}[k]$
    end
    //Post the updated row back to main memory
    $\texttt{post\_send}(\mathbf{J}_{local,j})$
    //Wait for all pending posts
    $\texttt{wait\_for\_completion}(\mathbf{J}_{local,j})$
    //Wait for next elimination row to arrive
    if $j + \texttt{spuid} \leq N$ **then**
      $\texttt{wait\_for\_completion}(\mathbf{J}_{local,j+\texttt{spuid}})$
    end
  end
  //Wait before starting next base row
  $\texttt{wait\_for\_notification}$
end

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



(a)



(b)

Figure 1. Performance of the parallel sparse direct solver in normalized GFlops: (a) Scaling with increasing number of SPUs and (b) Relative performance with dense and sparse solvers on a workstation for solving a 3264×3264 system.