Parallel solution for railway power network simulation

Y.F. Fung, T.K. Ho, W.L. Cheung
Department of Electrical Engineering, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
M.F. Ercan
Department of Electrical and Electronic Engineering, Singapore Polytechnic, Singapore

Abstract

The Streaming SIMD extension (SSE) is a special feature that is available in the Intel Pentium III and P4 classes of microprocessors. As its name implies, SSE enables the execution of SIMD (Single Instruction Multiple Data) operations upon 32-bit floating-point data therefore, performance of floating-point algorithms can be improved. In electrified railway system simulation, the computation involves the solving of a huge set of simultaneous linear equations, which represent the electrical characteristic of the railway network at a particular time-step and a fast solution for the equations is desirable in order to simulate the system in real-time. In this paper, we present how SSE is being applied to the railway network simulation.

1. Introduction

The Streaming SIMD Extensions (SSE) is a special feature available in the Intel Pentium III and P4 classes of microprocessors. The SSE can be considered as an extension to the MMX technology, which has been implemented by the Intel Pentium processors [1]. The MMX technology provides a set of 8 64-bit wide MMX registers and 57 instructions for manipulating packed data stored in the registers.

The major difference between SSE and MMX is in the data-type that can be operated upon in parallel. In MMX, special MMX registers, which are 64-bit wide, are provided to hold different types of data. However, it is limited to character (char), or integer values. On the other hand, the SSE registers are 128-bit wide and they can store floating-point values, as well as integers [2, 3]. There are eight SSE registers, each of which can be directly addressed using the register names [3]. Therefore, utilizing the registers is a straight-forward process with suitable programming tools. In the case of integers, eight 16-bit integers can be stored and processed in parallel. Similarly, four 32-bit floating-point values can be manipulated. Therefore, when two vectors of four floating-point values have been loaded into two SSE registers, as shown in Figure 1, SIMD operations, such as add, multiply, etc., can be applied to the two vectors in one single operation step.

Applications relying heavily on floating-point operations, such as 3D geometry, and video processing can be substantially accelerated [4]. Moreover, the support of floating-point values in the SSE operations has tremendously widened its applications in other problems including the railway power network simulation problem described in the following section.

2. Railway power network simulation

The major objective in railway power network simulation is to determine the electrical properties of the railway system. Components included in a railway network are substations, cables, and trains. Substations are positioned along the railway line and produces electrical power for the trains. Cables link between substations and is for electricity conduction. Trains move within the network; they draw current and act as electrical loads.

An electrified railway line is in fact a huge electrical circuit with the substations being the sources and the trains as moving loads (sources if the trains are using regenerative braking) [5]. The voltage seen by a train may vary with time and it will determine a train's traction performance, which in turn affects the train movement. Thus, it is necessary to attain the voltages at certain nodes, which are likely to be moving, of this electrical circuit at consecutive time intervals. A node of a network is defined as a point where two or more branches are connected together and current between two nodes should be constant throughout the distance at any time. This requires the basic circuit analysis and the solution of a matrix equation. The size of the coefficient matrix depends upon the number of nodes in the circuit, which is
largely determined by the number of trains on track. Inevitably, the matrix solution process is the most CPU consuming step in the simulation.

The matrix solution is in the form of:

\[ Ax = b \]  

(1)

where \( A \) is a symmetrical sparse matrix of order \( n \) representing the admittance linking the nodes, \( b \) represents the current produced by the source and \( x \) is an unknown solution vector defining the voltage attained by each node.

A common procedure (LU decomposition) [6] for solving (1) is to factor \( A \) into lower and upper triangular matrices \( L \) and \( U \) such that

\[ LUx = b \]  

(2)

and this then followed by forward/backward substitution of the form

\[ Lx' = b \]  

(3)

and

\[ Ux = x' \]  

(4)

Forward substitution first identifies the intermediate results \( x' \) and vector \( x \) is determined by backward substitution. Elements in the matrix \( A \) are being processed along the diagonal and on a row-by-row basis. Data stored in a row of the matrix can be processed in a group of four with the SSE registers and therefore shortening the computational time.

3. Programming with SSE

Programming with the SSE can be achieved by two different approaches. The SSE operations can be invoked by assembly codes embedded in a standard C/C++ programs. Alternatively, by utilizing the special data types we can develop a C/C++ program without any assembly coding. The new data type designed for the manipulation of the SSE operation is \( \text{F32vec4} \) [3]. It represents a 128-bit storage, which can be applied to store four 32-bit floating-point data. Similarly, there is also the type \( \text{F32vec8} \), which is used to store eight 16-bit values. These data types are defined as C++ classes and can therefore be applied in a C/C++ program directly.

In addition to the new data types, operations are derived to load, or pack, traditional data, such as floating-point values, into the new data structure. As an example, to load, or pack, four floating-point values into a \( \text{F32vec4} \), the function \( \text{mm_load}_\text{ps} \) can be applied. Once data are stored into the 128-bit data structure, functions that can manipulate the \( \text{F32vec4} \) type data can be called. This will result in parallel processing for two sets of four floating-point values. The function \( \text{mm_store}_\text{ps} \) is used to convert, or unpack, the data from the \( \text{F32vec4} \) type back to four floating-point values and the values are stored in an array.

4. Parallel LU decomposition based on SSE

The calculation involved in LU decomposition can be explained by the following equation:

\[
\begin{align*}
\text{For } k &= 0 \text{ to } n-2 \\
\text{Do} \\
\text{For } i &= k+1 \text{ to } n-1 \\
\text{Do} \\
\text{For } j &= k+1 \text{ to } n-1 \\
\text{Do} \\
& \quad a_{i,j} = a_{i,j} - \frac{a_{i,k} \times a_{k,j}}{a_{k,k}} \\
\end{align*}
\]  

(5)

In the above equation, \( a_{ij} \) represents elements in the \( A \) matrix.

According to (5), elements in the matrix \( A \) are being processed along the diagonal and in a row-by-row basis. Data stored in a row of the matrix map naturally into the \( \text{F32vec4} \) data and therefore, four elements in a row can be evaluated in a single step.

Based on (5), the term \( \frac{a_{i,k}}{a_{k,k}} \) is a constant when elements \( a_{k,k} \) in row \( i \) are being processed. It can, therefore, store in a \( \text{F32vec4} \) type value with the command \( \text{mm_load}_\text{ps} \). The command loads a single 32-bit floating-point value, copying it into all four words of the 128-bit storage. The pseudo codes shown in Figure 2 illustrate the steps performed in order to implement equation (5) using SSE functions.

In forward substitution, the operations can be represented by:

\[ x_i = b_i - \sum_{j=1}^{i-1} x_j \times L_{i,j} \]  

(6)

where \( x_i \) represents element in the \( \{x\} \) matrix as shown in equation (3);
\( b_i \) represents element in the \( \{b\} \) matrix
\( L_{ij} \) represents elements in the \( \{L\} \) matrix
SSE operations can be applied in the operation $x_j \cdot L_{ij}$.

Four elements of $x_j$ and $L_{ij}$ can be stored in two different $F32vec4$ data and multiplied at a single operation.

In backward substitution, the operations can be represented by

$$x_j - \sum_{m=j+1}^{m} x_m \cdot U_{jm} \over U_{jj}$$

where $U_{jj}$ represents elements in the Upper matrix $[U]$; $m$ is the size of the vector $[x]$. Similar to forward substitution, the multiplication of $x_m \cdot U_{jm}$ can be executed by SSE functions with four elements of $x_m$ and $U_{jm}$ being operated on at the same instead.

<table>
<thead>
<tr>
<th>Original</th>
<th>SSE</th>
<th>Speedup Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>431 ms</td>
<td>367 ms</td>
<td>1.17</td>
</tr>
<tr>
<td>91 ms</td>
<td>27 ms</td>
<td>3.37</td>
</tr>
</tbody>
</table>

Table 1: Timing results obtained from a railway network simulator

As described in the above paragraph, the data generated by the simulator are stored in a file, therefore, operations such as opening the file and reading data from the file must be performed. The file operations induce severe overhead in the results, as depicted in the first row of Table 2 (case 1). The results for case 2 are obtained by subtracting the overhead caused by the file operations from the total processing time. The speedup ratio derived from Case 2 is close to the ideal case of 4. Certainly, when the SSE algorithm is being incorporated into the simulator then the overhead due to the file operations will be significantly reduced and the efficiency of the simulator can be improved.

6. Conclusions

In this paper, we have presented the basic operations involved in utilizing the SSE features of the Pentium III processors. In order to examine the effectiveness of SSE, the railway network simulation problem was introduced and solved by applying SSE functions. According to our results, a speedup ratio around 3 can be obtained if the overhead induced by file operations can be minimized. This will be the case when the SSE algorithm is being embedded in the railway network simulator. The results are satisfactory because only minor modifications of the original program are needed in order to utilize the SSE features. Most importantly, additional hardware is not required for this performance enhancement. Therefore, SSE is a valuable tool for improving the performance of computation intensive problems, and the railway network simulation problem is one of the ideal applications.
However, there is also other form of overhead, which is induced by the SSE mechanism. This is substantiated by the results obtained from the randomly generated data, Table 1. The overhead is caused by the mechanism to pack data into, as well as unpack data from the 128-bit format. Further studies to identify the magnitude of such overhead should be carried out. In addition, the optimization of the available cache memory can also improve the overall performance and how the SSE algorithm can be further improved by optimizing the cache memory will be investigated in our future studies. Currently, only the DC railway system has been studied and we are also planning to apply the SSE approach for solving an AC system, which involves complex number arithmetic. As there is not a natural match between the SSE data types and complex number, and this will be a challenging task to determine a SSE based solution for the AC system.

References:


Acknowledgment:
This work is supported by The Hong Kong Polytechnic University and the Singapore Polytechnic.