EXPERIMENTAL ANALYSIS OF A MIXED-MODE PARALLEL ARCHITECTURE PERFORMING SEQUENCE SORTING

Samuel A. Finenberg, safineberg@iacen.uiowa.edu
Thomas L. Casavant, tom@eng.uiowa.edu
Parallel Processing Laboratory
Department of Electrical and Computer Engineering
University of Iowa
Iowa City, IA 52242 USA

Howard Jay Siegel, h@ecn.purdue.edu
Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907 USA

1. Introduction

This paper is a very brief summary of an algorithm and raw data gathered in empirically-based architecture research generated from experiments on a mixed-mode machine capable of operating in both SIMD and MIMD modes of parallelism. The experimentation is based on timing measurements made on the PASM [3] system prototype at Purdue utilizing carefully coded synthetic variations of a well-known algorithm. The algorithm used to measure and evaluate this system was based on bitonic sorting [1] of sequences stored in the PEs (processing elements, i.e., processor/memory pairs). This application has feasible solutions in both SIMD and MIMD modes of computation, as well as in two hybrids of SIMD and MIMD modes. Detailed explanation of the algorithm used, as well analysis and discussion of this results appears in [2].

2. Synthetic Algorithm Used

A variation on Batcher's basic bitonic sorting algorithm [4] is shown in Figure 1. Assume there are P=2p PEs (numbered 0 to P-1), the number of items to be sorted is N = 2p, and each PE initially contains a sorted list of N/P data items. In the algorithm, ADDR(i) is the jth bit of a PE's number, DTRin is the network input data transfer register, DTRout is the network output data transfer register, and cube means PEs whose numbers differ in i exchange data.

for i = 1 step 1 to p-1 do
  if ADDR(i) = 0 then type ← 0 else type ← 1
  for j = i-1 step -1 to 0 do
    for q = 1 step 1 to N/P do
      DTRin = X[q]
      cube
      Y[q] = DTRout
      merge(X[X',X'],X)
      choose(X'X,Y')
      Figure 1: Arbitrary sequence sorter.

In each PE, the portion of the list to be sorted is held in X, a one-dimensional array of N/P elements. For the q loop, each PE i receives a copy of the X array of PE cube(i), and stores it in Y. Then, in each PE, "merge(X,X',Y')" merges the sorted lists X and Y into the sorted list X,Y and places the lesser half in X' and the greater half in Y'. This is a simple O(N/P) "merge" routine for merging two sorted lists. It requires 3(N/P) if-then-else conditional operations. These are to check which of the elements being examined is lower, to check if the end of list X has been reached, and to check if the end of list Y has been reached. In MIMD mode, when the end of a list is reached, a PE simply copies the remainder of the other list onto the end of the new sorted sequence. In SIMD mode, PEs may reach the end of a list at different times. To handle this, the completed list is represented by an element with a value larger than all allowable values. This causes the remainder of the other list to be copied onto the end of the new sorted sequence.

After completing the "merge," the PEs use "choose(X,X',Y')" to choose between the two lists (pointers) depending on each PE's value of "type" and "ADDR."

choose(X,X',Y')
  if (type = ADDR(i))
    then X = X'
  else X = Y'

This work supported by the National Science Foundation under Grant Numbers CCR-8415456, CCR-8402900, by the Air Force Office of Scientific Research under Grant Number F49620-84-K-0006, by the Naval Ocean Systems Center under the High Performance Computing Block Grant, and in part by the NSF Software Engineering Research Center (SERC).

5. Experimental Results

As can be seen from Figures 2 and 3, the fastest version was clearly the S/MIMD version, with the execution time of the BMIMD version slightly greater at all points. After these two mixed-mode versions the MIMD version was next, with the SIMD version being the slowest. The p=2,3 cases produced identical data. This is due to the time complexity of the algorithm, which is proportional to (N/P)p(p+1)/2. This complexity is the same for P=4 and P=8: (N/P)p(p+1)/2 = (N/4)(2×3)/2 = (N/8)(3×4)/2.

MIMD vs. BMIMD

First, consider the MIMD and BMIMD versions. These were the most similarly implemented of the four versions. As can be seen from the graph, their execution times start out close and, as N increases, the difference between the execution times of MIMD and
the BMIMD versions increases. (Note that a constant vertical distance between lines of positive slope represents an increasing difference on this log scale.) The use of barrier synchronization is the only difference between the MIMD and BMIMD versions. The number of network transfers is \((N/P)(p+1)/2\), so as \(N\) increases the advantage gained by the more efficient network transfer in BMIMD mode also grows.

S/MIMD Comparison

The S/MIMD version was faster than the other versions and improved in relation to the MIMD and BMIMD versions as \(N\) increased. Its speed can be attributed to having almost no overhead necessary for network transfers, and the added advantage of the control flow overlap when in SIMD mode. In fact, the only overhead present in this version but not present in the others was that of explicitly transferring control between SIMD and MIMD modes. Notice that as \(N\) increases the advantage of S/MIMD over MIMD and BMIMD improves due to the \((N/P)(p+1)/2\) complexity of the \(q\) loop operations (see Figure 1) which benefit from SIMD control flow overlap and SIMD network transfers. Also, the mode changing overhead of S/MIMD mode is outside of the \(q\) loop and occurs only \(p(p+1)/2\) times. In BMIMD mode, the barrier synchronization is within the \(q\) loop, and therefore occurs \((N/P)(p+1)/2\) times.

SIMD Comparison

There is a large difference evident between the SIMD version and the other three versions. This can most readily be attributed to the design of the CCL. Note that the use of the CCL occurs within the \((N/P)(p+1)/2\) time complexity "merge" routine that was the source of both all of the PE local data conditional instruction overhead and most of the if-then-else serialization due to masking (as was mentioned in Section 2).

Effects of Condition Code Logic

Consider the execution times of the SIMD version both with and without conditional mask generation (Figures 2 and 3). While the shapes of these two curves are similar, there was a large difference between the performance of these versions. This difference is due to the overhead caused by PE mask generation. The SIMD version represents the current design of PASM's CCL, while the SIMD with no conditional mask generation (SIMD/no-CC) version represents a lower bound on execution time if conditional masks could be generated with zero cost. This data was collected by removing the relevant statements from the program. This is to avoid any time penalty caused by any inefficiencies in the specific implementation of data conditional masking in the prototype.

SIMD/no-CC vs. S/MIMD and BMIMD

As is evident from Figures 2 and 3, the SIMD/no-CC version is relatively close to the others for small \(N\), but its execution time increases sharply as \(N\) increases (recall execution time is on a log scale). In the S/MIMD and BMIMD versions, their execution increases smoothly and with a much smaller slope than the SIMD/no-CC version. This is mainly caused by the added serialization in the "merge" routine due to the three if-then-else statements it contains.

6. Conclusion

Experiments on the PASM prototype designed to examine the trade-offs among the SIMD, MIMD, BMIMD, and S/MIMD modes of parallelism were described. In addition, the effects of SIMD condition code logic were considered. Experiments consisted of measurements of the execution time of synthetic versions of a bitonic sorting algorithm. The bitonic algorithm has feasible implementations in each of the SIMD, MIMD, BMIMD, and S/MIMD modes, and further, exercises SIMD condition code logic, as well as network reconfiguration aspects of PASM. Execution times for different size lists, numbers of processors, and modes of parallelism were collected. This data was evaluated and discussed, examining the effects of these various parameters in the tests. It was also shown that a mixed-mode machine can utilize its mode switching to support SIMD/MIMD parallelism and hardware barrier synchronization abilities to improve its performance over both pure SIMD and pure MIMD modes. These abilities were exploited in the mixed-mode S/MIMD and BMIMD versions. Through the use of these mixed-mode techniques, the advantages of both the SIMD and MIMD modes were simultaneously exploited, and the advantage of a machine with this mode-switching capability for the given class of tasks was demonstrated. Further details are in [2].

Acknowledgments: The authors of this paper acknowledge many useful discussions with Ed Bronson, Wayne Nation, Pierre Pero, Tom Schwederski, and the other members of the Parallel Processing Laboratory Users Group (PPLUG).

References