Cache Conscious Matrix Transpositions and Parallel Implementation

Javier Hernandez
Advisor: Dr. Jaime Seguel
Mathematics Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
E-mail: jht_hedz@hotmail.com

Abstract

Time is important, nowadays. In computer programming time is crucial. The speed of processing large amount of data is today's major battleneck in several scientific and engineering applications. One solution of this problem could be to take advantage of the software and the hardware to improve the time required for processing data.

Maximizing the hardware capabilities and developing parallel algorithms are probably solutions to the problem of reducing the time of processing the information.

1. Introduction

Considering the existence of computers with more than one CPU (Central Processing Unit). How we can't get advantage of it? It parallel processing, just a mater of splitting a process into a number of processors? We can take advantage of them?

In this article, a parallel algorithm for transposing large matrices is designed. The algorithm is not only parallel but it is also cache conscious, this is to take advantage of the memory hierarchy of the machine. The algorithm was implemented in cilk5.2 and run on a symmetric multiprocessor with four CPUs.

2. The Algorithm

Transposing an N x M matrix is indeed a simple job. In serial environment, it would be just a matter of swapping memory allocation. The efficiency of such a serial transposition could however be degraded if the element to be swapped are not in cache. This will normally be true case if the matrix is large. The main idea behind our algorithm is to use a buffer that fits in cache to load sub-block of element in the matrix that are to be transposed. Perform then the swap in cache and overwrite the original matrix within the reallocated entries (see figure 1). This task can be performed in parallel since the sub-block loaded into the buffer are independent.

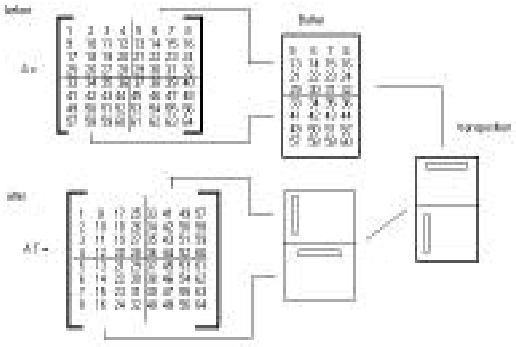


Figure 1

3. CILK Implementation

Cilk5.2 is a multithread extension of C developed by the Laboratory of Computers Science of the Massachusetts Instituted of Technology (MIT). Following the main guidelines provided by the authors of Cilk, we have developed a fully functional version in C of the above described algorithm. This C version is their altered by inserting the Cilk commands: cilk, spawn and sync. Parallelism is produced by the spawn command, which generates child procedures. The parent procedure that has spawned a child continues to execute until a sync command is reached (see figure 2).

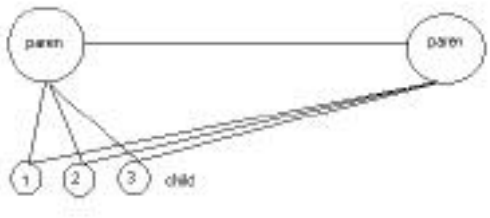


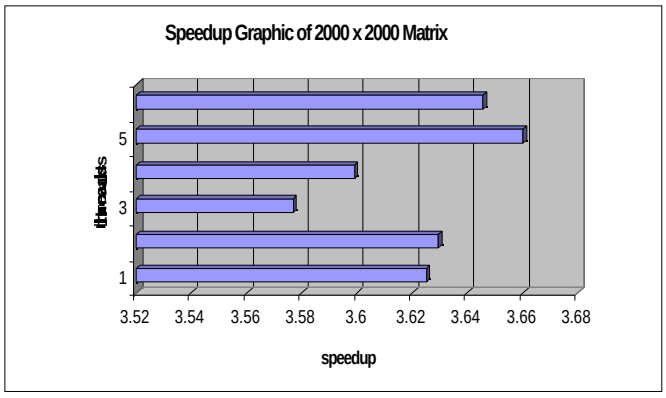
Figure 2

There are some excerpt of the code:

```
cilk void load_matrix_transp()
{
 int k,l,nk,ml;
 int count1=0, count2=0, count3=0;

 nk = n+alpha;
 ml = m+alpha;

 for(k=n; k < nk; k++)
  { count2=0;
    for(l=m; l < ml; l++)
     { B[count1][count2] = A[k][l];
       count2++;  }
    count1++;
    }
 count3 = alpha;
 for(k=n; k < nk; k++)
  { count2=0;
    for(l=m; l < ml; l++)
     { B[count3][count2] = A[l][k];
       count2++;  }
    count3++;
   }
  spawn transp_matrix();
}


cilk void diag_matrix()
{
 int tmp=0;
 int t,y;

  for(t=n; t < (n+alpha); t++)
   for(y=m; y <= t; y++)
    {
     if(t==y)
      {B[t][y];}
      else
        {
          tmp = B[t][y];
          B[t][y] = B[y][t];
          B[y][t] = tmp;
          }
     }
}
```

4. Statistics

The Cilk environment provides several statistics to measure the performance of the algorithm. The following chart (figure 3) shows the results of runs on a four processor symmetric multiprocessor, with 512Kb of L2 Cache and 400MHz Xeon processors.



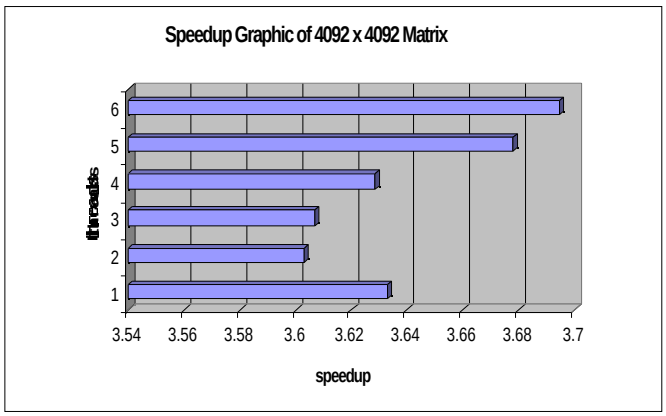


Figure 3

4. References

- Dr. Jaime Seguel, COMP5055 Parallel Computation, Class Notes 1999. UPR-Mayagüez
- Linear Algebra and Its Application. 2nd edition August 1999, by David C. Lay. Addison-Wesley Pub Co
- Cilk5.2 Reference Manual Supercomputing Technology Group MIT Laboratory For Computer Science. http://theory.les.mit.edu/~cilk