XMT-C and the Ear Decomposition Algorithm TJHSST Computer Systems Research Laboratory 2007-2008

Luis Alejandro Valentin

June 9, 2008

Abstract

This project takes the ear decomposition algorithm for partitioning maps and compares runtime efficiencies of a serial implementation and a parallel implementation, both of which are written in XMT-C. The number of nodes on the tested datasets span from 10 to 100. The number of edges in each span from the minimum possible to the maximum possible number of edges, where the minimum equals the number of nodes and the maximum equals $\binom{N}{2}$. The point at which the speed up of the parallel implementation equals the overhead lag is estimated.

Keywords: Parallel Programming, XMT-C, Ear Decomposition, Spawn

1 Introduction

Parallel programming is in no way a new concept. Unfortunately, for the past fifty years more emphasis has been put in improving run-time for serial programs. Now that hardware has almost hit its serial limit, it is turning to parallel implementations; dual cores, for example, are becoming more popular. More research will hopefully be put into parallel programming in the near future.

When serial algorithms do not parallelize well, new approaches are needed to tackle problems. In the case of the Depth-First Search, it does not convert to parallel well. The Ear Decomposition Search was created as the parallel equivalent of the DFS.

2 Background

2.1 PRAM

PRAM refers to an abstract machine for designing algorithms in parallel. This machine allows for an infinite number of processors that can each access the same memory in uniform time. The four PRAM models include:

1. EREW - explicit read, explicit write
2. CREW - concurrent read, explicit write
3. ERCW - explicit read, concurrent write
4. CRCW - concurrent read, concurrent write

The CRCW can then be subdivided into three types- common, arbitrary, and priority, depending on which processor writing is successful. The ERCR PRAM model is almost never considered for obvious reseasons. The first known implementation of CRCW PRAM is the the University of Maryland A. James Clark School of Engineering's ParaLeap prototype 64-core supercomputer.

2.2 XMT-C and ParaLeap

The language used on the ParaLeap is called XMT-C, eXplicit Multi-Threaded C. Simply, the language is C with three extra methods, spawn, ps, and psm. The spawn method allows the programmer to use multiple processors. The spawn method takes two arguements, start and end. Then, end-start+1 threads are run, each with an identification number from start to end, inclusive, that can be referenced from the dollar sign. While any number of processors can be called for, the computer only has 64 processors to run at any given moment. Figure 1 shows how the spawn method allows the programmer to move from a serial state to a parallel state. The ps method, short

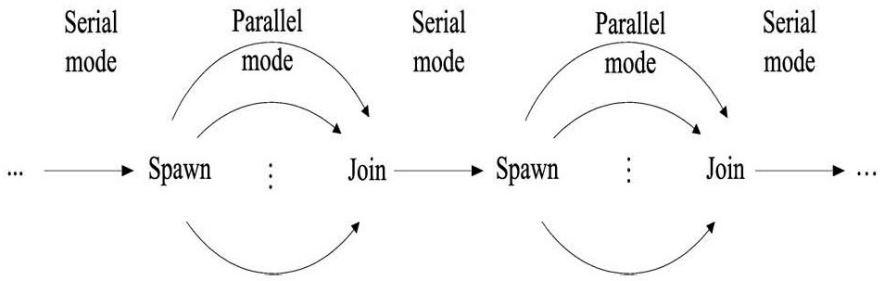


Figure 1: The ease of changing from parallel to serial mode[1]

for prefix sum, allows for different threads to communicate with each other. The ps method can only be called inside a spawn call. The ps method takes two arguements, step and base. Step is a local integer set at either 1 or 0. Base is a global variable of type psBaseReg. The ps method simultaneously sets the value of step equal to base and increments base by step. Refer to [2] for a definition of psm.[2]

ParaLeap is the nickname given to the University of Maryland A. James Clark School of Engineering's prototype supercomputer. The commands for compiling and running a program in XMT-C on this machine are xmtgcc32 and xmtfpga, respectively. [2]

2.3 Ear Decomposition

The Ear Decomposition algorithm is used for partitioning different maps into simple paths known as ears. Alone, the Ear Decomposition algorithm does not provide much information. Instead, it is used as an intermediate step for other algorithms. Some algorithms that use Ear Decomposition as an intermediate step include: [3][5]

1. st-numbering
2. search for tri-connected components
3. planarity testing

4. disjoint paths

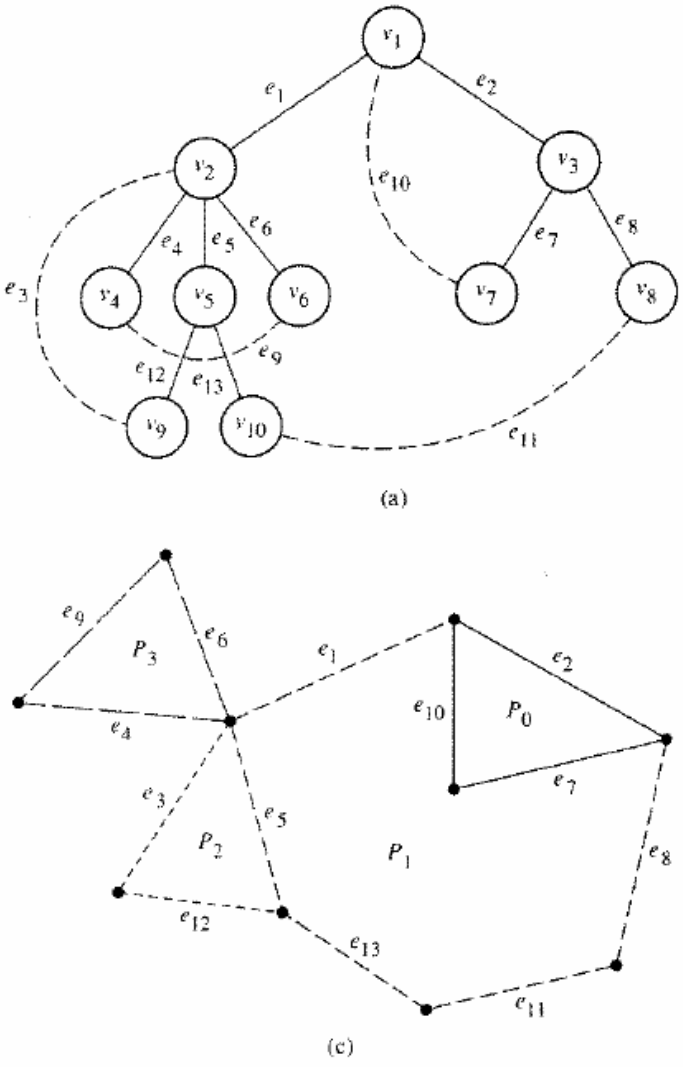
The ear decomposition algorithm is inputed a biconnected, or bridgeless, undirected graph G and outputs an array labeling each edge of G to the cooresponding ear. The ear decomposition algorithm follows the five steps below: [4]

1. Find a spanning tree T of G
2. Root T at an arbitrary vertex r, and compute $level(v)$ and $p(v)$, for each vertex $v \neq r$, where $level(v)$ and $p(v)$ are the level and the parent of v, respectively.
3. For each nontree edge $e = (u, v)$, compute $lca(e) = lca(u, v)$ and $level(e) = level(lca(e))$, where lca is the least common ancestor of nodes u and v. Set $label(e) := (level(e), s(e)$, where $s(e)$ is the serial number of e.
4. For each tree edge g, compute $label(g)$.
5. For each nontree edge e, set $P_e = e \bigcup g \in T | label(g) = label(e)$. Sort P_es by $label(e)$.

3 Procedures

Both the parallel and the serial Ear Decompositon algorithms were written in XMT-C and run on the ParaLeap. They follow the steps explained in section 2.1. The spanning tree was found using a Breadth First Search and was rooted at the node whose value was zero.

The twleve datasets tested had N nodes, where $N \in 10, 25, 50, 100$. For each N value, a dataset existed with E edges, where $E = 25$ and 75 percent of $\binom{N}{2}$, a completely connected dataset, and N, the fewest number of edges while maintaining biconnectivity. Input data was given in the form of four arrays, vertices, antiparallel, degrees, and a two dimensional edges array. Because random data is not garanteed to be bridgeless, a special program, makeData, was written to create the datasets. First, the program connects all node in a large ring, ensuring biconnectivity. Then, the program randomly adds edges until the specified number are included. Lastly, the numbers from makeData were run through memMapCreate32 in order to use in XMT environment.



Edge	Label
e_1	(0, 11)
e_2	(0, 10)
e_3	(1, 3)
e_4	(1, 9)
e_5	(0, 11)
e_6	(1, 9)
e_7	(0, 10)
e_8	(0, 11)
e_9	(1, 9)
e_{10}	(0, 10)
e_{11}	(0, 11)
e_{12}	(1, 3)
e_{13}	(0, 11)

(b)

Figure 2: (a) The spanning tree with with dashed edges representing non-tree edges. (b) An array showing edges and their cooresponding labels. (c) The original data showing ears P_0,P_1, P_2, and P_3. Image taken from page 242 of [4]

Each dataset was run three times and the clockcycles were averaged. The dataset with 50 nodes and 75 percent edges caused an unknown error, but can safely be assumed to run faster than its serial counterpart. Graphs and Tables of clockcycles can be found in Appendix A.

4 Results and Conclusions

From Figures 3,4, and 5, one can see that as the percent of edges increases, the number of nodes needed for the parallel program to outperform the serial version decreases. Conversely, as the number of nodes increases, the number of edges needed for the parallel program to outperform the serial version also decreases. This fact can be seen from Figures 5, 6, 7, and 8.

A more efficient Depth First Search can be used when finding a spanning tree to improve both parallel and serial runtimes. A special property of ParaLeap disproportionately puts serial programs to a disadvantage. When declaring an array, all values of the array begin as random numbers. Parallel algorithms take constant time to set these values to zero, whereas serial algorithms take linear time.

Further research should test values of N significantly greater than 64, the number of processors on ParaLeap, to view the full extent of the parallel speed-up. More research can be done to determine whether an increase in ears has an effect on runtime of the Ear Decompositon Algorithm.

References

[1] Caragea, George C., Bryant C. Lee, and Uzi Vishkin.Models for Advancing PRAM and Other Algorithms into Parallel Programs for a PRAM-On-Chip Platform. CRC Press, LLC, 2001.

[2] Caragea, George C., and Alexandros Tzannes. "XMT Toolchain Manual for XMTC Language, XMTC Compiler, XMT Simulator and XMT FPGA Computer". UMIACS. Version 2.1

[3] Ibarra, Louis, and Dana Richards. "Efficient parallel graph algorithms based on open ear decomposition." Parallel Computing 19 (1993): 873-886

[4] JaJa, Joseph. Introduction of Parallel Algorithms. Addison-Wesley, Mar 1992.

[5] Maon, Yael, Baruch Schieber, and Uzi Vishkin. "Parallel Ear Decomposition Search (EDS) and st-Numbering in Graphs." Theoretical Computer Science 47 (1986): 277-298

A Graphs

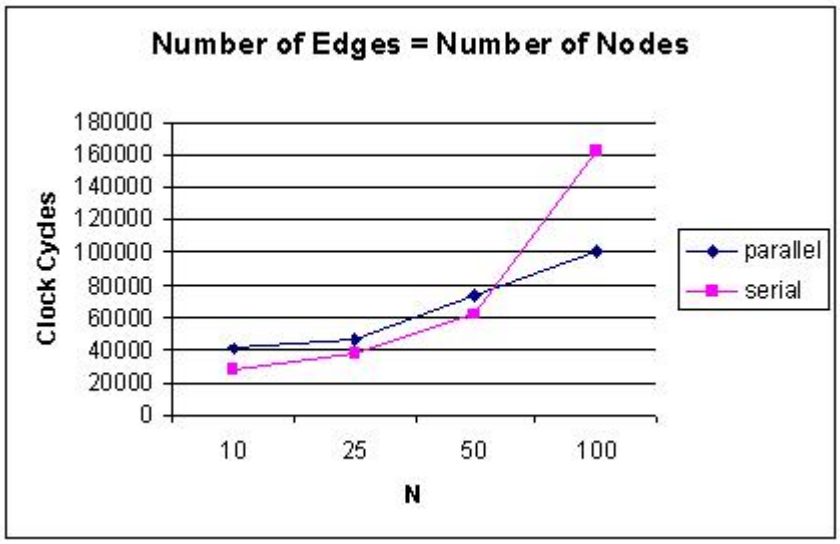


Figure 3: Number of Edges Held Constant

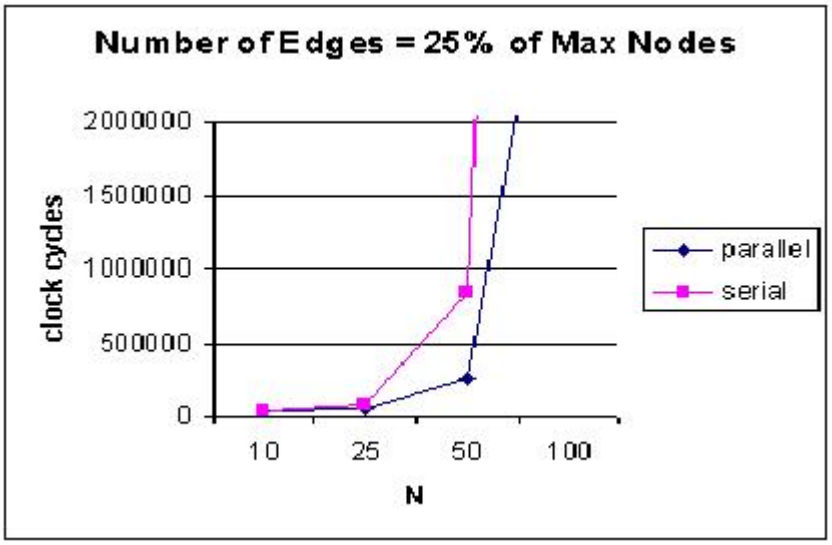


Figure 4: Number of Edges Held Constant

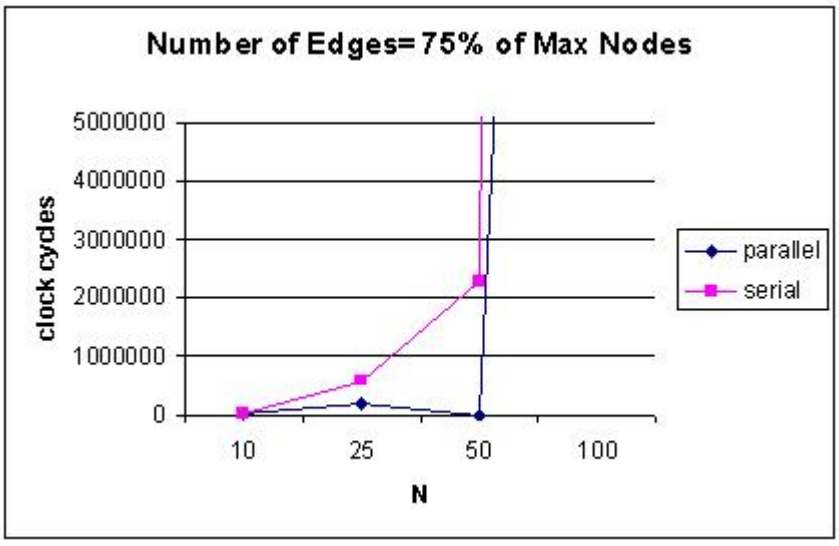


Figure 5: number of Edges Held Constant

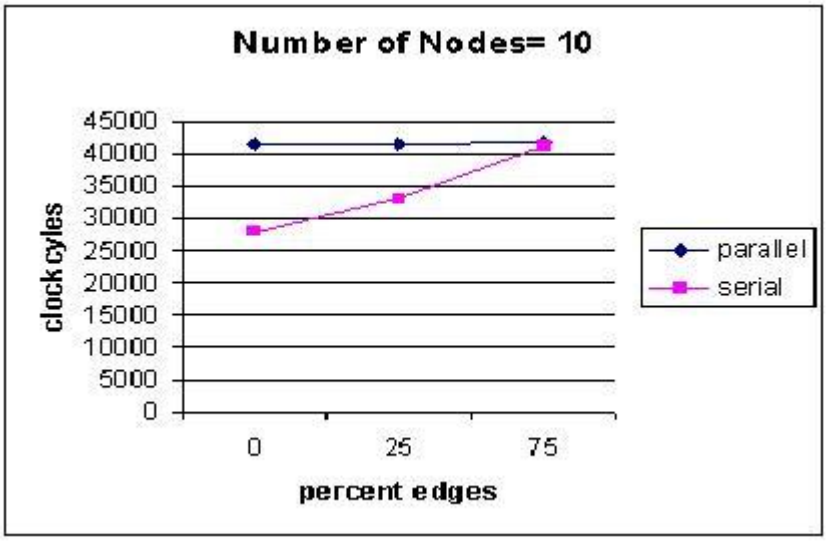


Figure 6: Number of Vertices Held Constant

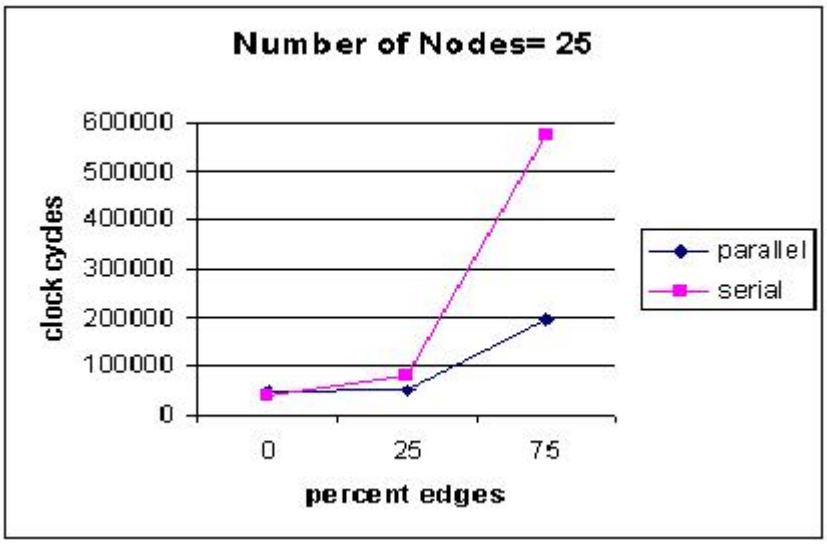


Figure 7: Number of Vertices Held Constant

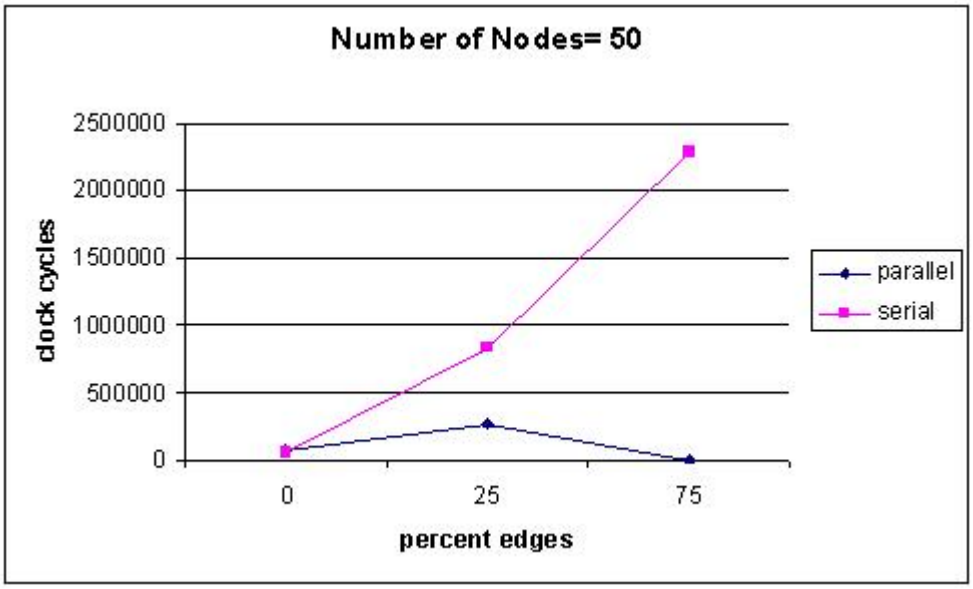


Figure 8: Number of Vertices Held Constant

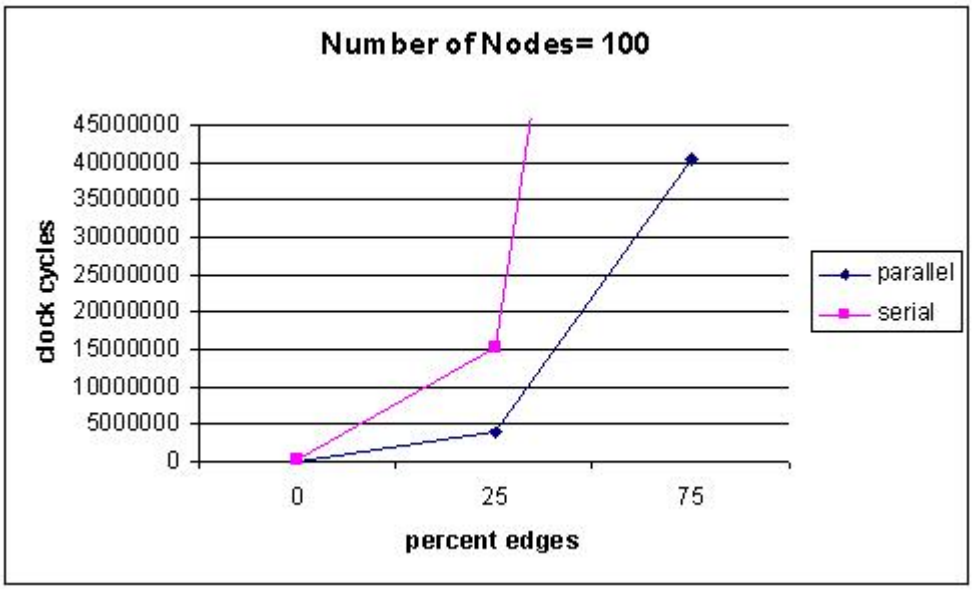


Figure 9: Number of Vertices Held Constant