SAND2011-0293C

Subgraph Isomorphism on a Multithreaded Shared Memory Architecture

Claire Ralph* Vitus J. Leung† William McLendon‡

January 11, 2011

Abstract

We adapt two deterministic subgraph isomorphism algorithms for the Cray XMT, a massively multi-threaded shared memory architecture. We expect this architecture to be especially suited for large graph matching problems. We compare a classical algorithm, Ullmann's, with a more recent one, VF2, which we adapt from the serial version.

1 Introduction

We compare deterministic algorithms for the subgraph isomorphism problem implemented for the Cray XMT. The Cray XMT is a massively multithreaded shared memory machine. The shared memory allows one to avoid having to partition the problem into multiple memory domains, thus making it ideal for large graph problems with no obvious partition scheme. The large number of threads on each chip allow the processor to work without stalls due to memory latency as long as enough streams are active. For large graph matching problems, the difficulty lies in effectively traversing a search space which consists of many instances which are largely independent of each other. We expect enough independent work (active streams) such that the processor is not stalled due to memory latency. This makes large graph matching problems great target applications for multithreaded machines.

*Cornell University, ccr53@cornell.edu
†Sandia National Laboratories, vjleung@sandia.gov
‡Sandia National Laboratories, wcmclen@sandia.gov

In this paper we run some experiments comparing two deterministic algorithms for the subgraph isomorphism problem. We eventually hope to find a high performing algorithm for inclusion in the MultiThreaded Graph Library (MTGL) [3]. We compare one of the earliest algorithms for this problem, known as Ullmann's algorithm [31], to the more recent algorithm known as the VF2 algorithm [9]. We expect Ullmann's algorithm to be highly parallelizable as mentioned in his paper. The VF2 algorithm, while originally developed for serial runs, has a similar structure to the Ullmann algorithm and thus we hope to adapt the algorithm to the multithreaded environment effectively.

Applications of subgraph isomorphism include cheminformatics [31], pattern discovery in databases [21], bioinformatics [28, 1], modeling social networks [30], computer-aided design [13, 27, 23], scene analysis [2], and robot vision [33].

2 Related Work

The deterministic exact subgraph isomorphsim problem is NP-complete [8]. However, certain cases of subgraph isomorphism may be solved in polynomial time [15]. Subgraph isomorphism is a generalization of the graph isomorphism problem which asks whether a graph G is isomorphic to a graph H [10, 29, 18, 24]. However, the complexity of graph isomorphism remains an open question [17]. Other related problems known to be NP-complete are induced subgraph isomorphism [17], maximum common subgraph [17, 5, 16], and graph edit distance [5, 6]. The randomize complexity of subgraph isomorphism is $\Omega(n^{3/2})$ [20]. The online version of the problem has also been studied [26].

The difficulty in the general case lies in the size of the search space which grows factorially with the size of the large graph. In order to explore this search space efficiently several techniques are used. We explore algorithms which attempt to prune the search space using what we refer to as "look ahead" rules which check if a candidate subgraph of G_B is consistent with G_A and thus may form part of a match [31]. For example if the candidate subgraph is made up of three nodes and $|G_A| = 5$ we know that none of the three nodes may have degree less than the node of smallest degree in G_A. The specific rules used in each algorithm are defined in the next section. Since these rules can be checked in parallel, they may be particularly suited to multithreaded parallelism.

Other classes of algorithms which attempt to make the exact problem tractable use techniques such as partitioning the large graph [4], specialized data structures [11, 9], or constraint satisfaction [22, 34]. Partitioning the large graph creates many smaller problems which can be handled through brute force enumeration. This has

shown to be effective in cloud computing, especially when the large graph has an underlying structure which allows for convenient partitioning [4], but the monolithic memory of multithreaded parallelism would seem to be at odds with such an approach. One of the algorithms we test uses some specialized data structures [9], and our results show that it is less effective. Since constraint satisfaction is difficult to implement in serial, it would only be more difficult in parallel.

Other techniques use heuristics which return sets of possible matches [32, 19, 12, 14, 25, 3]. These may return false positives [3], and thus the returned set needs to be checked for actual consistency. This can be done using brute force or possibly one of the algorithms we explore in this paper.

3 The Algorithms

We implemented two algorithms referred to as Ullmann [31] and VF2 [9] which we will define here. We first describe a brute force enumeration solution and then describe the techniques used to prune the search tree described in the brute force solution. One defines the fully enumerated search tree as follows. Enumerate the nodes of the small graph. Starting with the first node, we have $|V_B|$ possible matches from the large graph. This is the first level of the search tree. On the next level we match the second node of the small graph, v_2^A. We have $|V_B - 1|$ possibilities for each possible matching of the second node giving us $V_B * (V_B - 1)$ branches at the second level. We then match the third node of the small graph on the third level of the search tree. The complete search tree has depth V_A with $V_B * \ldots * (V_B - (V_A - 1))$ branches. This factorial growth at each level is what makes the problem intractable for graphs of nontrivial size.

We compare algorithms which attempt to prune this search tree as we explore it. In order to understand how the search space is truncated we first note that we can represent the search space we need to explore through a matrix we refer to as the M matrix which has dimension $|V_A| \times |V_B|$. Each row correlates with a vertex in G_A and each column with a vertex in G_B. An element of the matrix m_{ij} is 1 if vertex v_i^A can be mapped to v_j^B and 0 if not. For example if the degree of v_i^A is larger than the degree of v_j^B, we can set $m_{ij} = 0$. When we enumerate the search tree as we did in the brute force algorithm described above, at each level i which corresponds to finding a match in G_B for vertex v_i^A, we only include partial candidates such that $m_{ij} = 1$.

The first step in the algorithm is to set as many elements of the M matrix to zero using a degree test. One throws away all possible mappings where a vertex of G_A is mapped to a vertex of G_B with degree less than itself. After the first sweep

through all possible mappings, the degree of the vertices in G_B is updated to reflect the number of neighbors which still have the possibility of being mapped to a vertex in G_A. For instance let vertices $v, v' \in V_B$ be neighbors with degrees m and $m+1$ respectively where $m+1$ is the smallest degree of any vertex in G_A. On the first sweep, all mappings containing v will be thrown away but those containing v' will be kept. Prior to the second sweep the effective degree of v' will be updated to at most m as one of its neighbors v can not actually be used in the mapping. This means that on the second sweep, all mappings using v' will be thrown away. In addition to checking the degree of each vertex, we check that each of the neighbors of a vertex v_i^A can be mapped to a neighbor of v_j^B. If we find that a neighbor of v_i^A can not be mapped to a neighbor of v_j^B because the corresponding element of the M matrix is zero for all possible neighbors of v_j^B, we set $m_{ij} = 0$. This is repeated until a sweep occurs with no updates. As a matrix elements are only changed from 1 to 0, this can be done in parallel.

The next step is to start traversing the search tree which spans the truncated search space defined by our updated M matrix. The search space has now been refined and the second part of the algorithm begins. In this part of the algorithm we explore what is left of the search space. The simplest thing to do would be to enumerate all possible mappings left and check each one. We hope to prune the search tree however by using our look ahead rules. A partial candidate is a mapping which has been defined at some initial levels of the search tree. We may be able to determine that this partial candidate is not consistent with any possible solutions by using what we refer to as look ahead rules. For example if we are at level 2 and have assigned u_1 to v_m and now are looking at possible matchings for u_2. We know that any v_i assigned to u_2 must have degree greater than u_2. We call this a zero order rule. Let's say u_2 is a neighbor of u_1. Then we have the further constraint that v_i must be a neighbor of v_m. This will be called a nearest neighbor constraint as it depends on the nearest neighbors of u_2 and the neighbors of possible matching v. If we can rule out all possible matchings which assign the pair $(u_1, v_m), (u_2, v_i)$, then we do not need to explore that branch of the search tree any further. These constraints allow us to prune the search tree as we explore it. At each level, all partial candidates can be tested in parallel.

Ullmann developed the first algorithm of this type in 1976 [31]. Later Cordella, et al. published a similar algorithm known as VF2 [9]. Both algorithms use nearest neighborhood look ahead rules. The main difference between the two is that the Ullmann algorithm splits the neighbors into two types, those that have been matched and those that haven't. In the VF2 algorithm the group of nodes that have not been matched are further split into two groups. Those that have a neighbor in the

matching and those that don't. Thus far we have defined these rules in terms of an undirected graph, but they can be defined on a directed graph as well, which splits the above groups into smaller ones based upon whether a node is a predecessor or successor. In the directed version a neighbor node may be in more than one group.

The traditional implementation of the VF2 algorithm is optimized for a serial run and does not include the use of the M matrix. The checks performed in step one, where we sweep through the M matrix, are included in the consistency checks done in the second step. In our parallel implementation, we follow the algorithm defined by Ullmann and divide the VF2 algorithm into the two parts described above.

An additional difference between the algorithms should be noted. Traditionally the Ullmann algorithm solves the subgraph isomorphism problem defined in the introduction whereas the VF2 algorithm solves the induced subgraph isomorphism problem. In the induced subgraph problem we say a graph G_A is a subgraph only if it meets the requirements to be a subgraph in the problem defined above and only if an edge which exists between two vertices in G_B also exists between the corresponding vertices in G_A. Thus the induced subgraph problem is more restrictive than the traditional subgraph isomorphism problem and one expects to find less matches in many cases. This additional restriction may increase the speed of the code by increasing the pruning of the search tree or may slow the code by increasing the amount of testing that needs to be done in order to declare a match consistent. We did not explore this and instead adapted the VF2 code to solve the traditional problem rather than the induced subgraph problem in order to compare the benefits of using the more strenuous VF2 consistency rules over the ones defined by the Ullmann algorithm.

4 Implementation

Our implementation is based upon the one proposed in the 1976 Ullmann paper [31]. One copy of the M matrix is stored. For a given sweep through the M matrix, each element of the matrix is examined in parallel. The sweeps are done sequentially and continue until one occurs where the matrix is not updated.

In order to traverse the search tree, possible candidates are stored in an array with three indices. The first index defines the depth of the search tree at which we are. The second index corresponds to a candidate which contains a partial matching. The third index corresponds to a vertex in G_A which has been assigned a match in G_B. The value at that index is the id of the vertex in G_B. Memory is allocated at each depth which can hold all the partial candidates to be examined at that level. In order to keep track of the additional information needed by the VF2 algorithm

[9], we allocate an additional block of memory which keeps track of which nodes in G_B have been matched. This block of memory has the same structure as the one which keeps track of the partial candidates except that the third index corresponds to a vertex of G_B and the value at that index is the id of the matched vertex in G_A.

In order to perform the consistency checks, the Ullmann algorithm only needs the information contained in the partial candidate array. The VF2 algorithm needs the information contained in the partial candidate array and the reverse look up array. As we are doing the computation in parallel, we need to have all the candidates at the current level and at the level prior in memory simultaneously. This makes the memory requirements of the VF2 algorithm significantly larger than those required by the Ullmann algorithm. The VF2 algorithm also needs to know which group described in the algorithms section each node belongs too. We chose to recompute this information for each partial candidate as needed rather then store it so as to reduce memory costs. We could have done this with the reverse look up array as well. We plan to explore the cost in computation time versus the increase in memory requirements to do this on the fly computation in the future.

For both algorithms, all partial candidates at each level are checked for consistency in parallel. The levels, each of which corresponds to including a vertex in G_A in the matching, are looped through in serial. The ability to check all partial candidates at a given level in parallel is what gives these algorithms their power.

5 Experimental Results

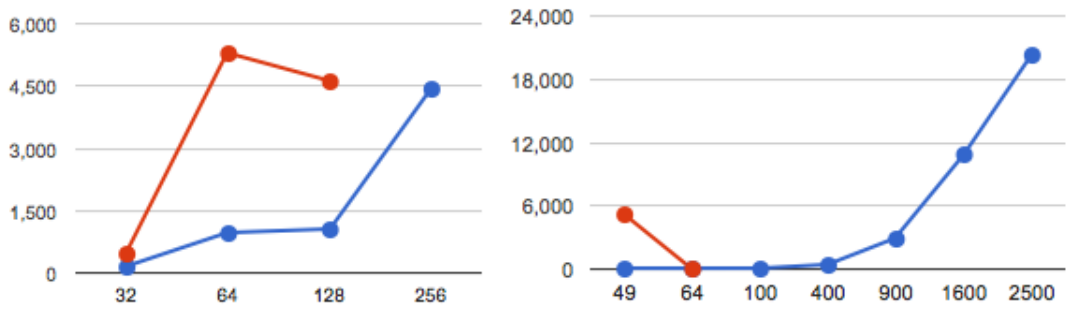

Figure 1: CPU Time (secs) vs. Num Vertices. Ullmann is in blue and VF2 in red. The left is data from RMAT graphs. The right is from mesh graphs.

We studied two algorithms, Ullmann [31] and VF2 [9] for the subgraph isomorphism problem. We tested the two algorithms on two types of graphs, 2D Mesh

Table 1: Total Memory References (Billions)

RMAT					Mesh						
Size	32	64	128	256	49	64	100	400	900	1600	2500
Ullmann	12.3	115	139	686	1.02	0.168	4.01	200	1529	10290	29500
VF2	43.8	607	486	-	674	1.4	-	-	-	-	-

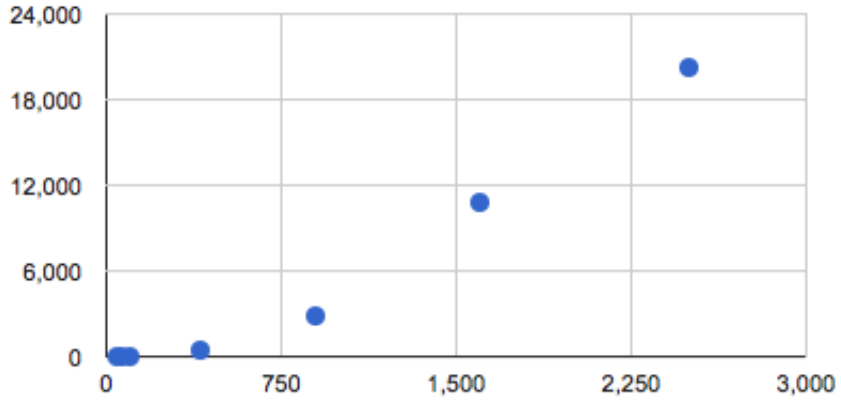

Figure 2: CPU Time (secs) vs graph size. Data was obtained from running Ullmann on various Mesh graphs

and RMAT [7]. The 2D Mesh graphs were generated using a library written in Python. The RMAT graphs were generated using the Multi Threaded Graph Library (MTGL). For each type of graph we created 7 graphs of various sizes. The target subgraph is of size 4 or 5 for all problem instances. The Ullmann algorithm outperformed the VF2 algorithm in all instances of the 2D mesh. Neither algorithm was able to solve the larger problems on the RMAT graphs in a reasonable amount of time. We believe this is due to the large amount of memory needed to store the possible candidates, which are numerous due to the density of the RMAT graphs we created. On the smaller instances which were solved, the Ullmann algorithm outperformed the VF2 algorithm in all cases. Unreported times for the VF2 algorithm indicates that the job was killed after exceeding the time used by Ullmann algorithm by at least 2 hours. A graph of CPU times is contained in Figure 1. Memory references are reported in Table 1. We also give a plot of CPU time vs. graph size for

Ullmann's algorithm run on a range of mesh graphs, Figure 2.

6 Discussion

The goal of this experiment was to explore whether the cost of using stricter consistency criteria during the second phase of an exact subgraph isomorphism algorithm outweighs the benefits of increased pruning of the search tree in a multithreaded environment. Our prelimary results suggest that this greater pruning of the search tree is not beneficial in a parallel environment even though it may be very beneficial in a serial environment. In a parallel environment, many partial candidates can be checked for consistency at the same time. One then might expect that the amount of work needed to do the consistency check at each level is what will dominate the run time rather than the number of candidates at each level. This seems to be supported by our results.

Another benefit of the Ullmann algorithm is the amount of memory necessary to store the partial candidates. Due to the nature of the look ahead checks employed in the VF2 algorithm, more information about each candidate must either be stored or computed on the fly. We took a middle ground approach in our implementation, storing some of this information and recomputing some of it as necessary. In our implementation, we expect this storage of extra information to double the memory requirements. Our observations during the experiments show the VF2 algorithm to use significantly more memory than the Ullmann algorithm consistently. This may be of concern in the future as all candidates at a given level must be stored in memory simultaneously.

In Figure 2 we plot CPU time versus graph size. We see that once we reach a certain problem size, the CPU time grows close to linear with problem size. This indicates a high level of parallelism. We do not expect to get near linear scaling in the small graph range due to the large amount of overhead needed to run jobs on the Cray XMT. Unfortunately we have not obtained similar results with the VF2 algorithm as we have not yet been able to solve problems of size sufficiently large.

7 Future Work

Our preliminary results suggest that strenuous look ahead checks are not beneficial in the multi-threaded environment. In the immediate future we thus plan to implement a stripped down version of the Ullman algorithm which uses even lighter consistency checks. We expect this to increase the number of candidates checked at each level, but

to decrease the amount of time necessary for each check for each partial candidate. As all candidates at a given level can be checked in parallel and the number of levels is bounded by the size of the target graph, we hope to see a decrease in run time.

Another possibility for increasing performance involves the coupling of front end heuristics to our deterministic algorithm. As mentioned in the related work section, one of these heuristics has already been implemented for the Cray XMT and is part of the MTGL.

Other natural extensions of this work include adapting our algorithms to handle attribute graphs and exploring inexact matching algorithms implemented for the Cray XMT. These algorithms have many applications, for example, enzymatic pathway matching in the field of Biology.

Acknowledgements

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under Contract No. DE-AC04-94AL85000. C. C. Ralph thanks the DOE Computational Science Graduate Fellowship for funding.

References

[1] Peter J. Artymiuk, Helen M. Grindley, Andrew R. Poirrette, David W. Rice, Elizabeth C. Ujah, and Peter Willett. Identification of β-sheet motifs, of ψ-loops, and of patterns of amino acid residues in three-dimensional protein structures using a subgraph-isomorphism. *Journal of Chemical Information and Computer Science*, 34(1):51–64–62, 1994.

[2] Ricardo Baeza-Yates and Gabriel Valiente. An image similarity measure based on graph matching. In *Proc. 7th Int. Symp. on String Processing and Information Retrieval*, pages 28–38. IEEE Computer Science Press, 2000.

[3] Jonathan W. Berry, Bruce Hendrickson, Simon Kahan, and Petr Konecny. Software and algorithms for graph queries on multithreaded architectures. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–14, 2007.

[4] Matthias Bröcheler, Andrea Pugliese, and V. S. Subrahmanian. Cosi: Cloud oriented subgraph identification in massive social networks. In *International*

Conference on Advances in Social Network Analysis and Mining (ASONAM), pages 248–255, 2010.

[5] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[6] H. Bunke. Error-correcting graph matching: On the influence of the underlying cost function. *IEEE Transactions Pattern Analyis and Machine Intelligence*, 21(9):917–922, 1999.

[7] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *Proc. 4th SIAM Int'l Conf. Data Mining*, pages 442–445. SIAM Press, 2004.

[8] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[9] Luigi P. Corella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[10] D. G. Corneil and C. G. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the Association for Computing Machinery*, 17(1):51–64, 1970.

[11] J. Cortadella and G. Valiente. A relational view of subgraph isomorphism. In *Proc. 5th Int. Seminar on Relational Methods in Computer Science*, pages 45–54, 2000.

[12] Andrew D. J. Cross, Richard C. Wilson, and Edwin R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.

[13] C. Ebeling and O. Zajicek. Validating vlsi graph layout by wirelist comparison. In *Proc. of the Conference on Computer Aided Design (ICCAD)*, pages 172–173, 1983.

[14] Yasser El-Sonbaty and M. A. Ismail. A new algorithm for subgraph optimal isomorphism. *Pattern Recognition*, 31(2):205–218, 1998.

[15] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms and Applications*, 3(3):1–27, 1999.

[16] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance measure combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753–758, 2001.

[17] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[18] Georg Gati. Further annotated bibliography on the isomorphism disease. *Journal of Graph Theory*, 3:95–109, 1979.

[19] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[20] Hans Dietmar Gröger. On the randomized complexity of monotone graph properties. *Acta Cybernetica*, 10(3):119–127, 1992.

[21] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *1st IEEE Interational Conference on Data Mining*, pages 313–320, 2001.

[22] Javier Larrosa and Gabriel Valiente. Graph pattern matching using constraint satisfaction. In *Proc. Joint APPLIGRAPH and GETGRATS Workshop and Graph Transformation Systems*, pages 189–196, 2000.

[23] Zong Ling and David Y. Y. Yun. An efficient subcircuit extraction algorithm by resource management. In *Proc. of 2nd International Conference on ASIC*, pages 9–14, 1996.

[24] B. T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recogn.*, 32(12):1979–1998, 1999.

[25] Bruno T. Messmer and Horst Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.

[26] Bruno T. Messmer and Horst Bunke. Efficient subgraph isomorphism detection: A decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[27] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *Proceeding of the 30th Design Automation Conference*, pages 31–27, 1993.

[28] N. Pržulj, D. G. Corneil, and I. Jurisica. Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics*, 22(8):974–980, 2006.

[29] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

[30] Tom A. B. Snijders, Philippa E. Pattison, Garry L. Robins, and Mark S. Handcock. New specifications for exponential random graph models. *Sociological Methodology*, 36(1):99–153, 2006.

[31] Julian R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[32] Richard C. Wilson, Adrian N. Evans, and Erwin R. Hancock. Relational matching by discrete relaxation. *Image and Vision Computing*, 13(5):411–421, 1995.

[33] E. K. Wong. Model matching in robot vision by subgraph isomorphism. *Pattern Recognition*, 25(3):287–303, 1992.

[34] Stephane Zampelli, Yves Deville, Christine Solnon, Sebastion Sorlin, and Pierre Dupont. Filtering for subgraph isomorphism. In *Principals and Practices of Constraint Programming (CP)*, pages 728–742, 2007.