CLA 2011, Amedeo Napoli, Vilem Vychodil, *Eds.*, p413–416, Nancy, France, 17-21 Oct,
ISBN 978-2-905267-78-8

Formal Concept Analysis on Graphics Hardware

W. B. Langdon, Shin Yoo, and Mark Harman

CREST centre, Department of Computer Science,
University College London Gower Street, London WC1E 6BT, UK

Abstract. We document a parallel non-recursive beam search GPGPU FCA CbO like algorithm written in nVidia CUDA C and test it on software module dependency graphs. Despite removing repeated calculations and optimising data structures and kernels, we do not yet see major speed ups. Instead GeForce 295 GTX and Tesla C2050 report 141 072 concepts (maximal rectangles, clusters) in about one second. Future improvements in graphics hardware may make GPU implementations of Galois lattices competitive.

Keywords: software module clustering, MDG, close-by-one, arithmetic intensity

1 Introduction

Formal Concept Analysis [7] is a well known technique for grouping objects by the attributes they have in common. It can be thought of as discrete data clustering. In general the number of conceptual clusters grows exponentially. However there are a few specialised algorithms which render FCA manageable, even on quite large problems, provided the object-attribute table is sparse [10]. Krajca, Outrata and Vychodil [10] report considerable improvement in FCA algorithms in the last two decades. All these successful algorithms use depth first tree search to find all the conceptual clusters in an object-attribute table.

Computer graphics gaming cards (GPUs) are relatively cheap and yet offer far more computing power than the computer's CPU alone. (E.g. a 295 GTX contains 480 fully functioning processors and yet costs only a few hundred pounds.) Also microprocessor trends suggest faster computing will require parallel computing in future. There are already hundreds of millions of computers fitted with graphics hardware which might be used for general purpose computing [3].

Krajca *et al.* [10] report using a distributed computer to overcome the "major drawback [of FCA's] computational complexity". They report their parallel algorithm PCbO gives near linear speed increase with number of computing nodes in a network of up to 15 PCs. In other work [11] they conclude that there is no universal best FCA data structure. Instead they suggest that the optimum performance will depend upon the application. In earlier work, Huaiguo Fu had created a parallel implementation of NextClosure but it was limited to 50 attributes [5] but this was subsequently greatly extended [6]. However, like Krajca *et al.* [10], both Fu's [5] and [6] approaches use conventional distributed computers composed of a few CPUs rather than hundreds of GPU processing elements.

Similarly Djoufak Kengue *et al.* [4]'s ParCIM implementation used a conventional network of 8 computers connected in a star fashion with MPI. Ours is the first FCA implementation to run in parallel on computer graphics cards (GPUs).

2 CUDA FCA Implementation

Although In-close [1] claims to be faster we easily obtained FCbO [9] from Source Forge. We initially implemented the Krajca sequential algorithm [9] in Python. This was followed by a version in CUDA C, where `ComputeClosure` is implemented in parallel on the GPU. (For details see our technical report [13].)

Krajca's routines `ComputeClosure` and `GenerateFrom` essentially form a depth first search algorithm which builds and navigates a tree of formal concepts from a binary 0/1 matrix describing which object has which property. Since the search is recursive and operates on one point in the tree at one time, it is unsuitable for parallel operation on graphics cards. Our graphics card parallel version retains the tree but uses beam search rather than depth first search.

Instead of proceeding to the first leaf of the tree, recursively backing up and then going forward to the next leaf and so on, in beam search, we also start from the top of the tree and then proceed along every branch to the next level. This requires saving information on the beam for every node at that level. Beam search next expands the search again to cover everything at the next level and so on until all the leafs of the tree have been reached. Notice instead of working on a single point in the tree the beam covers many points which can be worked on in parallel. Indeed within a couple of levels we can get a beam containing tens of thousands of individual search points which can be processed independently. This suits the GPU architecture which needs literally thousands of independent processing threads for it to deliver its best performance [12]. You will have spotted that in an exponential problem, like FCA, beam search quickly runs out of memory.

Even for quite modest tree depths the beam width is limited by the available space in the GPU card. (We have a configuration limit of 1.8 million simultaneous parallel operations.) When a beam search exceeds this limit, only the first 1.8 million searches are loaded onto the GPU and the rest of the beam is queued on the host PC. (Although we have not done this, in multi-GPU systems it would be possible to split the beam between the GPUs, allocating up to 1.8 million to each GPU.) The GPU only searches to the next level. It returns the concepts found by the searches and the newly discovered branches which remain to be searched. The concepts are printed by the host PC and the new branches are added to the end of the beam to await their turn. Effectively the beam becomes a queue of points in the tree waiting to be searched. The number of parallel searches is mostly limited by the need to have space on the GPU for all the potential new branches. This depends upon the tree's fan out which is problem dependent. Nonetheless the GPU can manage modest real software engineering examples (e.g. dependence clustering of the Linux kernel). Notice the beam will contain a mixture of pending search points at different depths in the tree.

Table 1. Performance on, FCA benchmarks, random module dependency graphs, and Software Engineering datasets [8]. Time given in seconds, except longest Python run which is hours:mins:secs. (For $\frac{1}{2}$ 295 GTX and Tesla C2050 the total time on the GPU is given.)

Dataset	Size	Density	Concepts	FCbO	Python	295 GTX	C2050
krajca	5×7	54%	16	0.00	0.11	0.01	0.01
wiki	10×5	44%	14	0.00	0.03	0.00	0.00
random	10×10	20%	16	0.00	0.04	0.00	0.00
random	100×100	2%	137	0.00	0.40	0.02	0.01
random	200×200	2%	420	0.00	4.33	0.00	0.01
random	500×500	2%	2861	0.01	162.60	0.02	0.02
bison	37×37	24%	692	0.00	0.32	0.00	0.01
compiler	33×33	6%	24	0.00	0.05	0.00	0.00
dot	42×42	28%	1302	0.00	0.71	0.00	0.01
grappa	86×86	7%	850	0.00	2.54	0.01	0.01
incl	172×172	2%	238	0.00	1.84	0.00	0.01
ispell	24×24	34%	432	0.00	0.15	0.01	0.01
linuxConverted	955×955	2%	141072	0.73	15:42:51	1.79	0.93
mtunis	20×20	29%	110	0.00	0.05	0.00	0.01
rcs	29×29	37%	1074	0.00	0.46	0.01	0.02
swing	413×413	2%	3654	0.01	208.71	0.03	0.02

3 Results

FCbO (version 2010/10/05) was downloaded and compiled without changes on a 2.66 GHz PC with 3 Gigabytes of RAM running 64 bit CentOS 5.0. The performance of FCbO, our Python code and our CUDA code on two types of GPU are given in Table 1. They show performance on: two bench mark problems, a selection of randomly generated symmetric object-attribute pairings and software module dependency graphs of real world example programs.

4 Discussion

It is unclear why our code does not do better.

We would expect a linear speed advantage for FCbO from both using 64 bit operations and from using compiled rather than interpreted code. However on sizable examples, the ratio between the speed of FCbO and that of our Python code is huge. This hints that FCbO has some algorithmic advantage.

GPUs are often limited by the time taken to move data rather than to perform calculations. "Arithmetic intensity" is the ratio of calculations per data item. Typically this is in the range 4–64 FLOP/TDE [2, p206], we estimate the arithmetic intensity of Krajca *et al.*'s algorithm [9] is less than 1. Thus a potential problem might be there is simply is not enough computation required by FCA compared to the volume of data.

Newer versions of CUDA have make it easier to overlap GPU operations. However our implementation does not do this. Since the work is spread across the multi-processors, we suspect that idle time is not a major problem.

5 Conclusions

There are many problems which are traditionally solved by depth first search. However this may not suit low cost computer graphics GPU hardware. We have implemented a form of beam search and demonstrated it on several existing FCA benchmarks and ten software engineering dependence clustering problems [8]. GPU beam search may also be more widely applicable.

Acknowledgements

I am grateful for the assistance of Gernot Ziegler of nVidia. Steve Worley, Sarnath Kannan, Stephen Swift, Stan Seibert and Yuanyuan Zhang. Software engineering MDGs were supplied by Spiros Mancoridis. Tesla donated by nVidia. Funded by EPSRC grant EP/G060525/2.

References

1. S. J. Andrews. In-close, a fast algorithm for computing formal concepts. In *Conceptual Structures Tools Interoperability Workshop at the 17th International Conference on Conceptual Structures*, Moscow, 26-31 July 2009.
2. M. Christen, O. Schenk, and H. Burkhart. Automatic code generation and tuning for stencil kernels on modern shared memory architectures. *CSRD*, 26(3):205–210.
3. B. Del Rizzo. Dice puts faith in nvidia PhysX technology for Mirror's Edge. NVIDIA Corporation press release, Nov 19 2008.
4. J. Djoufak Kengue, P. Valtchev, and C. Tayou Djamegni. Parallel computation of closed itemsets and implication rule bases. In I. Stojmenovic, *et al.*, eds., *ISPA 2007*, *LNCS 4742*, pp359–370. Springer.
5. Huaiguo Fu and E. Nguifo. A parallel algorithm to generate formal concepts for large data. In P. Eklund, ed., *ICFCA*, *LNAI 2961*, pp141–142. Springer, 2004.
6. Huaiguo Fu and M. O'Foghlu. A distributed algorithm of density-based subspace frequent closed itemset mining. In *HPCC*, pp750–755. IEEE, 2008.
7. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, 1999.
8. M. Harman, S. Swift, and K. Mahdavi. An empirical study of the robustness of two module clustering fitness functions. In H.-G. Beyer, *et al.*, eds., *GECCO 2005*.
9. P. Krajca, J. Outrata, and V. Vychodil. Parallel recursive algorithm for FCA. In R. Belohlavek and S. O. Kuznetsov, eds., *CLA 2008*, Olomouc, Czech Republic.
10. P. Krajca, J. Outrata, and V. Vychodil. Parallel algorithm for computing fixpoints of Galois connections. *Ann Math Artif Intel*, 59:257–272, 2010.
11. P. Krajca and V. Vychodil. Comparison of data structures for computing formal concepts. In V. Torra, *et al.*, eds., *MDAI 2009*, *LNCS 5861*, pp114–125. Springer.
12. W. B. Langdon. Graphics processing units and genetic programming: An overview. *Soft Computing*, 15:1657–1669, Aug. 2011.
13. W. B. Langdon, S. Yoo, and M. Harman. Non-recursive beam search on GPU for formal concept analysis. RN/11/18, Computer Science, UCL, London, UK, 2011.