D. Dryml: Parallel algorithms for FCA using graphical hardware

Department of Computer Science, Palacky University, Olomouc (17. listopadu 12, CZ–77146 Olomouc, Czech Republic)

Introduction

The world of supercomputers has been changing drastically in the last few years. It has been shifting from clusters of general purpose CPUs to combination of the CPUs with an aid of less versatile but much powerful graphical cards (GPGPU). The graphical cards can process data in highly parallel Single Instruction Multiple Data (SIMD) fashion, for example to apply one operation simultaneously to hundreds of memory/array cells. One of fields that can benefit from GPGPU are operations of linear algebra. For example matrix multiplication, addition, subtraction etc. It is possible to compute simultaneously hundreds of fields of product matrix with gain of 2 to 24 times faster compared to normal Octave implementation of the operations [7]. This poster aims on usage of GPGPU for computation of Formal Concept Analysis (FCA).

It is known that computing number of formal concepts from formal context is P-complete [6]. That means that it can't be computed in logarithmic space (we need whole context to compute all concepts) and it also means that it is hard to construct efficient parallel or distributed algorithm for this purpose. This fact is a logical result of concept lattice being an ordered set usually without ordering being complete a.i. for two different concepts c_1 and c_2 the infima or suprema usually isn't c_1 nor c_2. This means that even if we split recursive execution of the algorithm [2] we will most probably compute some of the concepts multiple times from multiple (parallel) execution sequences.

Previously there has been work on implementation of crisp FCA using Nvidia's CUDA GPGPU framework [4] but there has not been significant improvements over FCbO [5]. In my opinion and also as mentioned in the paper [4] there are two main reasons for lack of improvement:

:: usage of bit operations - PCbO with this approach computes 32 or even 64 operations $\wedge, \vee$ in one processor tick and also representation of necessary data shrinks by the factor of 32/64. Thus in most cases all necessary data fit to processor cache negating necessity to access much slower RAM.

:: overhead associated with GPGPU computations - to compute something on GPU it is necessary to allocate memory on it than transfer data to it, compile the GPU program (kernel) to be executable, execute the kernel and than write computed data back to the main RAM of computer.

These two reasons make GPGPU computations on crisp contexts faster only for big contexts (even mushroom isn't large enough). On the other hand it starts to be faster for much smaller fuzzy contexts and even for the size 64x64 there is noticeable speedup.

Formalizaton

:: data table: $\langle X, Y, I \rangle$, where $I \subseteq X \times Y$

:: X is set of objects; Y set of attributes; $I : X \times Y \to L$

:: $I(x, y)$ is degree to which attribute y applies to object x

:: concept-forming operators:

$$A^{\uparrow}(y) = \bigwedge_{x \in X} (A(x) \to I(x, y)),$$
$$B^{\downarrow}(x) = \bigwedge_{y \in Y} (B(y) \to I(x, y)).$$

:: formal (fuzzy concept $\langle A, B \rangle \in L^X \times L^Y$ such that $A^{\uparrow} = B$ and $B^{\downarrow} = A$

:: set of all concept-clusters: $\mathcal{B}(X, Y, I)$

:: concept hierarchy:

$$\langle A_1, B_1 \rangle \leq \langle A_2, B_2 \rangle \text{ iff } A_1 \subseteq A_2 \text{ (iff } B_2 \subseteq B_1)$$

:: $\mathcal{B}(X, Y, I)$ with $\leq$ yields concept lattice

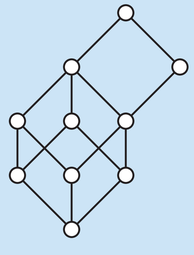
:: Łukasiewicz pair of adjoint operations

$$a \otimes b = max(a + b - 1, 0)$$
$$a \to b = min(1 - a + b, 1)$$

:: Gödel pair of adjoint operations

$$a \otimes b = min(a, b)$$
$$a \to b = \begin{cases} 1 \text{ if } a \leq b, \\ b \text{ otherwise.} \end{cases}$$

	y_1	y_2	y_3	y_4	y_5
x_1	1	0.5	0.5	1	1
x_2	1	1	1	1	0.5
x_3	0	0	0.5	0.5	1



Algorithm

For sake of comparison I used GenerateFuzzyConcepts algorithm [3] that is partial (attributes are crisp) fuzzy modification of CbO algorithm as described in [2] to compute fuzzy formal concept lattices.

```
GenerateFuzzyConcepts(⟨A, B⟩, y, Y)
process ⟨A, B⟩ (e.g., print on screen);
if ⟨A, B⟩ is minimal concept or y > |Y| then
  return;
end
for j form y to |Y| do
  if B(j)! = 1 then
    C = A ∩ Y(y)↓;
    D = C↑;
    skip = false;
    for k form 1 to j − 1 do
      if B(k) ∩ Y_j! = D(k) ∩ Y_j then
        skip = true;
        break;
      end
    end
  end
  if skip == false then
    GenerateFuzzyConcepts(⟨C, D⟩, j + 1);
  end
end
```

It is possible to compute at least two operations/kernels on GPU. First computes $C = A \cap Y(y)^{\downarrow}$ the intent of new concept. Second that computes Galois connection $C^{\uparrow}$ the extent of new concept. Both of these operations/kernels can be computed on multiple objects or attributes at the same time and thus benefit from OpenCL.

Dataset	Size	Normal (a)	OCL (b)	a / b
IPAQ	4510x16	35123	117980	0.2977
IPAQ'	16x4510	2505085	1291444	1.940
Amusic	900x26	43372	109442	0.3963
Amusic'	26x900	395	572224	0.6901
random	64x64	19502	93796	0.208
random	128x128	59609	108642	0.549
random	256x256	171786	174666	0.9835
random	512x512	1224660	340006	3.6019

Results show that it is necessary to have big matrix to benefit from OpenCL. It is partially due to Java (another API) and partially due to that intersection program/kernel isn't intensive enough to justify its usage (API overhead). On the other hand ↑ shows its power as OpenCL shows the speedup on the contexts with big number of attributes.

Future Research

:: Implement OpenCL fuzzy CbO algorithm in pure C. The results from Java are influenced by another layer of translation from Java OpenCL API to C OpenCL API and back. During tests of feasibility of OpenCL the first results showed that pure C implementation of finding all attribute concepts OpenCL started to benefit from context of size 64x64 and the speedup grew with the growing context.

:: Determining if a usage of GPU is possible and beneficial for other fuzzy FCA algorithms such as FCbO.

:: Implementation and measurements of PCbO with usage of multiple queues or GPU's.

:: Determining a minimal size of two-valued formal context that starts to benefit from GPGPU usage.

:: Implicitly-parallel implementation of linear algebra functions to Octave that would use GPGPU implicitly in case it would be beneficial for the sake of running program. As the Octave is vector and matrix oriented language it would make a perfect candidate for such kind of research. Moreover, higher languages such as Octave are beneficial for the sake of quicker development time and even pure encapsulation of OpenCL interface in Octave would make OpenCL usage much easier and quicker than in pure C.

References

[1] Belohlavek R., De Baets B., Outrata J., Vychodil V.: Computing the lattice of all fixpoints of a fuzzy closure operator. IEEE Transactions on Fuzzy Systems 18(3)(2010), 546-557.

[2] Krajca, P. and Outrata, J. and Vychodil, V.: Parallel recursive algorithm for FCA. CLA (2008), 71-82.

[3] Martin, T. and Majidian, A.: Finding fuzzy concepts for creative knowledge discovery. International Journal of Intelligent Systems 28(1)(2013), 93-114.

[4] Langdon, W. B. and Yoo, S. and Harman, M.: Formal Concept Analysis on Graphics Hardware. CLA (2011), 413-416.

[5] Krajca, P. and Outrata, J. and Vychodil, V.: Advances in Algorithms Based on CbO. CLA (2010), 325-337.

[6] Kuznetsov, S. O.: Learning of simple conceptual graphs from positive and negative examples. Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, (1999), 384-391.

[7] Bosi, LB. and Mariotti, M. and Santocchia, A.: GPU Linear algebra extensions for GNU/Octave. Journal of Physics: Conference Series 368(1)(2012), 325-337.

International Center for Information and Uncertainty
SSIU 2014: International Spring School and Workshop "Information and Uncertainty"
http://mcin.upol.cz/SSWIU-2014/

esf european social fund in the czech republic
EUROPEAN UNION
MINISTRY OF EDUCATION, YOUTH AND SPORTS
OP Education for Competitiveness
INVESTMENTS IN EDUCATION DEVELOPMENT