Ontology-based Knowledge Representation for PORPLE

[CSC766 Final Report]

Feifei Wang
North Carolina State University
fwang12@ncsu.edu

Yue Zhao
North Carolina State University
yzhao30@ncsu.edu

Xipeng Shen
North Carolina State University
xshen5@ncsu.edu

ABSTRACT

Data placement is important for the performance of a GPU (Graphic Processing Unit) program. However, where to place the data is a complex decision for a programmer to make. Among the recent techniques in solving data placement problem, PORPLE is a representative one. PORPLE is a portable data placement engine that uses hardware information (memory systems and processors) described by memory specification language (MSL) and software information (data access patterns) gathered from a compiler called PORPLE-C. Because of the two different representations, it is hard to share common understanding of the program, and reuse information. Providing a more general, uniform and reusable representation can make data replacement decisions more efficient, interoperable and reusable.

In this paper, we apply ontology-based techniques to systematically and formally represent both hardware information and software information used by PORPLE hoping to achieve efficiency, interoperability and reusability. Specifically, we transform the information of GPU memory systems and processors, and the data access patterns gathered by PROPLE-C to ontology which can be used by PORPLE for data replacement.

General Terms

Compiler

Keywords

Compiler, Ontology, ...

1. INTRODUCTION

Data placement is essential for the performance of a GPU (Graphic Processing Unit) program [**?**]. However, where to place the data depends on the hardware information of the GPU and software information of the program and its input. The hardware information of the GPU includes its memory systems and processors, while the software information means the data access patterns associated with the input to the programs. The memory systems of GPUs are becoming increasingly complex. For example, there exists more than eight types of memory (including caches) on the Tesla M2075 GPU. These memories have different size limitations, block sizes, access constraints and etc. Also, the suitable placements depend on the program inputs since different inputs to a program may lead to different data access patterns, and thus require different data placement. As a result, data placement problem is difficult but should be solved.

There have been some efforts to address the data placement problem [**?**, **?**, **?**, **?**]. Among them, PORPLE [**?**] is a representative one since it considers various types of GPU programs. PORPLE is a portable data placement engine that takes both hardware information (memory systems and processors) and software information (data access patterns) into consideration and use them to make data placement decisions. PORPLE obtains information about memory systems and processors from memory specification language (MSL), and uses the runtime profiling to acquire the data access patterns. In such a sense, PORPLE uses two different types of representations, which makes it hard to share common understanding of the program and reuse information. Providing a more general, uniform and reusable representation can improve the efficiency, interoperability and reusability of PORPLE and potentially some other work.

There exists various techniques to represent knowledge [**?**]. Recently, ontology-based knowledge bases are becoming increasingly popular. Ontology is a general-purpose modeling for knowledge resources used to define a common vocabulary and a shared understanding explicitly [1, **?**]. It has been successfully used to build knowledge bases in many fields [**?**, **?**, **?**, **?**]. Motivated by their advances, we choose ontology as the representation.

In this paper, we apply ontology-based techniques to systematically and formally represent both hardware information and software information hoping to make PORPLE more efficient, interoperable and reusable. Specifically, we transform the information of GPU memory systems and processors, and the data access patterns gathered to ontology which can be used by PORPLE for data replacement. Note that although our work is applied to PORPLE, it can also be applied to other work.

This paper is organized as follows. In Section 2, we present the motivation of our work. Section 3 illustrates the challenges of the project, our solutions, and lessons we learned. In Section 4 we explain our methodology in detail. Section 5 shows the results. Section 6 concludes this paper and dis-

CSC766 '15 Spring Raleigh, North Carolina USA

cusses some possible future work.

2. MOTIVATION

This work is motivated by an observation that PORPLE uses two different patterns for hardware information and software information.

This work is also motivated by some previous work of using Ontology to systematically and formally accumulate, represent, reuse, and manipulate knowledge about software, hardwar,e aoptimizations and so on.... For example, OpenK [**?**] adapts ontology-based techniques for building open, extensible, and reusable knowledge bases to systematically and formally accumulate, represent, reuse, and manipulate knowledge about HPC software, hardware, optimizations and so on.

3. CHALLENGE, SOLUTION AND LESSONS

The main challenge I encountered is to understand ontology, and I use .. to solve .. The lessons I learned ...

3.1 Challenge

3.2 Solution

3.3 Lessons

4. METHODOLOGY

5. RESULTS

6. CONCLUSION AND FUTURE WORK

It adapts to inputs and memory systems; it allows easy extension to new memory systems; it requires no offline training; in most cases, it optimizes data placement transparently with no need for manual code modification.

PORPLE has three advantages. extensibility- GPU architecture changes rapidly, and every generation manifests some substantial changes in the memory system design. For a solution to have its lasting values, it must be easy to extend to cover a new memory system. Our solution features MSL (memory specification language), a carefully designed small specification language. MSL provides a simple, uniform way to specify a type of memory and its relations with other pieces of memory in a system. GPU memory has various special properties: Accesses to global memory could get coalesced, accesses to texture memory could come with a 2-D locality, accesses to shared memory could suffer from bank conflicts, accesses to constant memory could get broadcast, and so on. (use list to occupy more space. :D)

Second, the solution should be input-adaptive. Different inputs to a program could differ in size and trigger different data access patterns, and hence demand different data placement. Since program inputs are not known until runtime, the data placement optimizer should be able to work on the fly, which entails two requirements.

Third, the solution should have a good generality. Data placement is important for both regular and irregular GPU programs. A good solution to the data placement problem hence should be applicable to both kinds of programs.

Ontology [1, **?**] is a general-purpose modeling for knowledge resources. It uses precise descriptive statements about knowledge of some domain. It is designed to represent rich and complex knowledge about things, groups of things, and relations between things... provide mutual understanding ...

Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the W3C. (W3C OWL 2 Web Ontology Language) OWL is a computational logic-based language to express ontologies. Knowledge expressed in OWL can be reasoned with by programs either 1. to verify the consistency of that knowledge or 2. to make implicit knowledge explicit.

An important feature of PORPLE is its capability to be easily extended to cover new memory systems. We achieve this feature by MSL. In this section, we first present the design of MSL, and then describe a high-level interface to enable easy creation of MSL specifications.

A. MSL MSL is a small language designed to provide an interface for compilers to understand a memory system.

Figure 2 shows its keywords, operators, and syntax written in BackusNaur Form (BNF). An MSL specification contains one entry for processor and a list of entries for memory. We call each entry a spec in our discussion. The processor entry shows the composition of a die, a TPC (thread processing cluster), and an SM.

Each memory spec corresponds to one type of memory, indicating the name of the memory (started with letters) and a unique ID (in numbers) for the memory. The name and ID could be used interchangeably; having both is for conveniences. The field âĂIJswmngâĂİ is for indicating whether the memory is software manageable. The data placement engine can explicitly put a data array onto a software manageable memory (versus hardware managed cache for instance). The field âĂIJrwâĂİ indicates whether a GPU kernel can read or write the memory. The field âĂIJdimâĂİ, if it is not âĂIJ?âĂIJ, indicates that the spec entry is applicable only when the array dimensionality equals to the value of âĂIJdimâĂİ. We will use an example to further explain it later. The field after âĂIJdimâĂİ indicates memory size. Because a GPU memory typically consists of a number of equal-sized blocks or banks, âĂIJblockSizeâĂİ (which could be multi-dimensional) and the number of banks are next two fields in a spec. The next field afterwards describes memory access latency. To accommodate access latency difference between read and write operations, the spec allows the use of a tuple to indicate both. We use âĂIJupperLevelsâĂİ and âĂIJlowerLevelsâĂİ to indicate memory hierarchy; they contain the names or IDs of the memories that sit above (i.e., closer to computing units) or blow the memory of interest. The âĂIJshareScopeâĂİ field indicates in what scope the memory is shared. For instance, âĂIJsmâĂİ means that a piece of the memory is shared by all cores on a streaming multiprocessor. The âĂIJconcurrency FactorâĂİ is a field that indicates parallel transactions a memory (e.g., global memory and texture memory) may support for a GPU kernel. Its inverse is the average number of memory transactions that are serviced concurrently for a GPU kernel. As shown in previous studies [11], such a factor depends on not only memory organization and architecture, but also kernel characterization. MSL broadly characterizes GPU kernels into compute-intensive and memory-intensive, and allows the âĂIJconcurrencyFactorâĂİ field to be a tuple containing two elements, respectively corresponding to the values for memory-intensive and compute-intensive kernels. We provide more explanation of âĂIJconcurrencyFactorâĂİ

through an example later in this section, and explain how it is used in the next section.

GPU memories often have some special properties. For instance, shared memory has an important feature called bank conflict: When two accesses to the same bank of shared memory happen, they have to be served serially. But on the other hand, for global memory, two accesses by the same warp could be coalesced into one memory transaction if their target memory addresses belong to the same segment. While for texture memory, accesses can benefit from 2-D locality. Constant memory has a much stricter requirement: The accesses must be to the same address, otherwise, they have to be fetched one after one.

How to allow a simple expression of all these various properties is a challenge for the design of MSL. We address it based on an insight that all these special constraints are essentially about the conditions for multiple concurrent accesses to a memory to get serialized. Accordingly, MSL introduces a field âĂIJserialConditionâĂİ that allows the usage of simple logical expressions to express all those special properties. Figure 3 shows example expressions for some types of GPU memory. Such an expression must start with a keyword indicating whether the condition is about two accesses by threads in a warp or a thread block or a grid, which is followed with a relational expression on the two addresses. It also uses some keywords to represent data accessed by two threads: index1 and index2 stand for two indices of elements in an array, address1 and address2 for addresses, and word1 and word2 for the starting addresses of the corresponding words (by default, a word is 4-byte long). For instance, the expression for shared memory, âĂIJblockword1âL'ă word2 word1

B. Example To better explain how MSL offers a flexible and systematic way to describe a memory system, we show part of the MSL specification of the Tesla M2075 GPU in Figure 4 as an example. We highlight two points. First, there are three special tokens in MSL: the question mark âĂIJ?âĂİ indicating that the information is unavailable, the token âĂIJomâĂİ indicating that the information is omitted because it appears in some other entries, the token âĂIJnaâĂİ indicating that the field is not applicable to the entry. For instance, the L2 spec has a âĂIJ?âĂİ in its banks field meaning that the user is unclear about the number of banks in L2. PORPLE has some default value predefined for each field that allows the usage of âĂIJ?âĂİ for unknowns (e.g., 1 for the concurrencyFactor field); PORPLE uses these default values for the unknown cases. The L2 spec has âĂIJomâĂİ in its upperLevels and lowerLevels fields. This is because the information is already provided in other specs. The L2 spec has âĂIJnaâĂİ in its dim field, which claims that no dimension constraint applies to the L2 spec. In other words, the spec is applicable regardless of the dimensionality of the data to be accessed on L2.

Second, some memory can manifest different properties, depending on the dimensionality of the data array allocated on the memory. An example is texture memory. Its size limitation, block size, and serialization condition all depend on the dimensionality of the array. To accommodate such cases, an MSL spec has a field âĂIJdimâĂİ, which specifies the dimensionality that the spec is about. As mentioned earlier, if it is âĂIJnaâĂİ, that spec applies regardless of the dimensionality. There can be multiple specs for one memory that have the same name and ID, but differ in the âĂIJdimâĂİ and other fields.

In this example, the concurrency factors of global and texture memory are set to 0.1 for memory-intensive GPU kernels and 0.5 for compute-intensive GPU kernels. They are determined based on a prior study on GPU memory performance modeling [11]. To determine a kernel is compute or memory intensive, we measure the I PCduring the profiling phase by checking performance counters (explained in Section VI). A kernel with I PC smaller than 2 is treated as memory-intensive, and compute-intensive otherwise.

MSL simplifies porting of GPU programs. For a new GPU, given the MSL specification for its memory system, the PROPLE placer could help determine the appropriate data placements accordingly. (don't forget figures)

introduce the data access pattern here...

Components of OWL

Ontologies

Individuals represent objects in the domain in which we are interested Properties are binary relations3 that link two individuals together Classes are used to model abstract knowledge for grouping objects with similar characteristics.

Classes can be organized into superclass-subclass hierarchy and they are described or defined by the relationships that individuals participate in.

GPU data placement

Model MSL (Memory Specification for Extensibility) - translate grammar-based rules into ontologies

Modeling and matching memory access patterns - static program analysis of CUDA program, online profile is possible by embedding OWL reasoning engines

reference: âĂIJOWL 2 Web Ontology Language Primer (Second Edition).âĂİ Accessed March 21, 2015. http://www.w3.org/TR/2 owl2-primer-20121211

A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3

http://www.semanticweb.gr/thea/

7. RELATED WORK

PORPLE contains three key components: a specification language MSL for providing memory specifications, a compiler PORPLE-C for revealing the data access patterns of the program and staging the code for runtime adaptation, and an online data placement engine Placer that consumes the specifications and access patterns to find the best data placements at runtime. These components are designed to equip PORPLE with a good extensibility, applicability, and runtime efficiency. Together they make it possible for PORPLE to cover a variety of memory, handle both regular and irregular programs, and optimize data placement on memory on the fly.

Ontology is widely used. A lot of work has been done. Moor et al. [**?**] and Leenheer et al. [**?**] focus on community-based evolution of knowledge-intensive systems with Ontology.

A lot of work has been done by the Semantic Web community on formalizing, reasoning and querying ontologies.(!!!)

Tang et al. [**?**] implement a profile compiler that support ontology-based, community-grounded, multilingual, collaborative group decision making by Ontology engineering to lift terms in multilingual sources to the conceptual level in order to tackle the problems of ambiguity and misunderstanding.(!!!)

Also, Ontology is one of the hottest topic in software en-

Table 1: Correctness for All the Questions

Classes and Instances	ClassAssertion()
Class Hierarchies	SubClassOf()
Object Properties	ObjectPropertyAssertion()
Property Hierarchies	SubObjectPropertyOf()
Datatypes	DataPropertyAssertion()

Table 2: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

gineering. For example, create of Web-portals on the basis of ontology and use ontology for navigation in information arrays. for intellectualizing software agents. [**?**] by Kleshche himself. point out that ..

The potential of ontology is more than above.

8. THE *BODY* OF THE PAPER

8.1 Citations

Citations to articles [**?**, **?**, **?**, **?**], conference proceedings [**?**] or books [**?**, **?**] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [**?**]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[**?**].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed or supported.

8.2 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

Figure 1: A sample black and white graphic (.eps format).

Figure 2: A sample black and white graphic (.eps format) that has been resized with the `epsfig` command.

8.3 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

8.4 Theorem-like Constructs

Figure 4: A sample black and white graphic (.ps format) that has been resized with the `psfig` command.

Table 3: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 3: A sample black and white graphic (.eps format) that needs to span two columns of text.

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TEX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TEX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LATEX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

9. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LATEX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

10. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

11. REFERENCES

[1] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5):907–928, 1995.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure

within an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.