論文 / 著書情報
Article / Book Information

題目(和文)	GPU上のオブジェクト指向プログラミングのメモリ効率化
Title(English)	Memory-Efficient Object-Oriented Programming on GPUs
著者(和文)	シュプリンガー マティアス
Author(English)	Matthias Springer
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第11319号, 授与年月日:2019年9月20日, 学位の種別:課程博士, 審査員:増原 英彦,遠藤 敏夫,首藤 一幸,南出 靖彦,脇田 建
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第11319号, Conferred date:2019/9/20, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Category(English)	Doctoral Thesis
種別(和文)	論文要旨
Type(English)	Summary

（博士課程）
Doctoral Program

論 文 要 旨

THESIS SUMMARY

専攻： Department of	Mathematical and Computing Science 専攻	申請学位（専攻分野）： Academic Degree Requested	博士 Philosophy Doctor of
学生氏名： Student's Name	Matthias Springer	指導教員（主）： Academic Supervisor(main)	Prof. Hidehiko Masuhara
		指導教員（副）： Academic Supervisor(sub)	

要旨（英文 800 語程度）
Thesis Summary（approx.800 English Words ）

High-performance, general-purpose GPU computing has long been a tedious job, requiring programmers to write hand-optimized, low-level programs. Such programs are hard to develop, debug and maintain. Even though object-oriented programming (OOP) is well established in other domains and appreciated for its good abstraction, expressiveness, modularity and developer productivity, it is regarded as too inefficient for high-performance computing (HPC). This is despite the fact that many important HPC applications exhibit an inherent object structure.

Our goal is to bring fast object-oriented programming to SIMD architectures, especially GPUs. We show that an object-oriented programming style is feasible on GPUs without sacrificing performance. We investigate how GPU parallelism can be expressed with object orientation and how object-oriented code that runs on GPUs can be optimized. In the course of this thesis, we present four libraries/prototypes that allow programmers to utilize OOP on GPU.

The first library, Ikra-Ruby, is a Ruby library for high-level, array-based GPU programming. Programmers express parallelism over an array with small, functional, customizable operations such as ``parallel map" or ``parallel stencil". Programmers can utilize OOP to compose a larger GPU program from multiple parallel operations in a modular way, forming a computation graph, similar to many machine learning frameworks. The code inside parallel operations is simple and usually not object-oriented. Ikra-Ruby optimizes the computation graph by fusing multiple parallel operations into a small number of CUDA kernels. We show that this kernel fusion process can be expressed as a type inference pass.

To explore OOP within parallel GPU code, we developed a C++/CUDA library Ikra-Cpp. While Ikra-Ruby provides various parallel operations, we found that a simpler model with only one type of operation, ``parallel do-all", is sufficient for a many applications. We discovered a broad object-oriented programming model that can be implemented efficiently on massively parallel SIMD accelerators. We call this model Single-Method Multiple-Objects (SMMO), because parallelism is expressed by running a method on all objects of a type (parallel do-all). SMMO fits well with the data-parallel execution pattern of GPUs and has many important real-world applications such as simulations for population dynamics, evacuations, wildfire spreading, finite element methods or particle systems. SMMO can also express breadth-first graph traversals and dynamic tree updates/constructions.

OOP is slow on SIMD architectures mainly because of inefficient memory access. Getting data into and out of vector registers is often the biggest bottleneck and peak memory bandwidth utilization can be achieved only with efficient vector transactions. To optimize the memory access of Ikra-Cpp applications, we developed an embedded C++ DSL that stores objects of the same type in the well-known Structure of Arrays (SOA) data layout. SOA is a form of structure splitting that stores all values of a field together. SOA is a standard optimization and best practice for GPU programmers but without proper language support, it leads to less readable code and breaks language abstractions. Ikra-Cpp allows programmers to experience the performance benefit of SOA without sacrificing code readability or OOP abstractions in C++.

As the main research contribution of this thesis, we extended Ikra-Cpp with DynaSOAr, a fully-parallel, lock-free, dynamic memory allocator. DynaSOAr improves the usage of allocated memory with an SOA data layout and achieves low memory fragmentation through efficient management of free and allocated memory blocks with lock-free, hierarchical bitmaps. Contrary to other state-of-the-art allocators, our design is heavily based on atomic operations, trading raw (de)allocation performance for better overall application performance. In our benchmarks, DynaSOAr achieves a speedup of SMMO application code of up to 3x over state-of-the-art allocators. Moreover, DynaSOAr manages heap memory more efficiently than other allocators, allowing programmers to run up to 2x larger problem sizes with the same amount of memory.

In a system with dynamic memory allocation, the efficiency of an SOA data layout depends heavily on low memory fragmentation. If data is fragmented, more vector accesses are needed for the same number of bytes, reducing the benefit of SOA. To further improve memory fragmentation, we extended DynaSOAr with CompactGpu, an incremental, fully-parallel, in-place memory defragmentation system. CompactGpu defragments the heap by merging partly occupied memory blocks. We developed several implementation techniques for memory defragmentation that are efficient on SIMD/GPU architectures, such as finding defragmentation block candidates and fast pointer rewriting based on bitmaps.

In this thesis, we show that, contrary to common belief, object-oriented programming is feasible on GPU without sacrificing performance, if the programming system provides well-designed abstractions and programming models that can be implemented efficiently on GPUs. SMMO is the main programming model throughout this thesis and we show through various examples that it is sufficiently expressive and that it can run efficiently on GPUs. While this thesis focuses mostly on memory access performance, future work could focus on more advanced OOP features such as virtual function calls or advanced modularity constructs such as multiple inheritance.

備考 ： 論文要旨は、和文 2000 字と英文 300 語を 1 部ずつ提出するか、もしくは英文 800 語を 1 部提出してください。
Note : Thesis Summary should be submitted in either a copy of 2000 Japanese Characters and 300 Words (English) or 1copy of 800 Words (English).