Multicore, Multithreaded, and/or Multi-GPU-Kernel VSIPL Standardiization, Implementation, and Programming Impacts: Syntax, Semantics, Models

Anthony Skjellum, PhD

RunTime Computing Solutions, LLC
1500 1st Avenue North, Suite C112/U19, Birmingham, AL 35203, USA
tony@runtimecomputing.com

Overview

There is immediate need to define, refine, support, and standardize performance-portable VSIPL/VSIPL++ profiles with potential additional API and well-defined augmented semantics (behaviors and rules) because of the rapid adoption of fine-grain SMP and multicore processors and/or multi-simultaneous-kernel GPUs, some or all of which can now comprise a computational hierarchy in which VSIPL/VSIPL++ must execute. VSIPL providers needed such standards to successfully support their customers. Pragmatically, this paper begins by describing threaded user-level programming models for VSIPL that need to be standardized now and are already needed/requested by existing VSIPL users on existing and near-term systems (*e.g.*, VxWorks or Linux for PPC 8641D, Intel Core i7, and soon, e500-based massively multicore processors as well), including the basic use cases requested by actual defense-program adopters of these standards. The concepts described here apply both to VSIPL and VSIPL++ formulations, and are largely orthogonal to considerations of distributed memory parallelism already considered in the VSIPL++ standard or the concurrent use of VSIPL and MPI, DRI, or other message-passing language.

Contributions

This paper makes the following contributions:

- This work will inform the key body of VSIPL and VSIPL++ adopters to the new dimensions of concurrency and how they both enable and challenge these standards as well as the user programs that use VSIPL or VSIPL++ at present.
- A review of the thread-safe aspects of the existing VSIPL standard is provided, with comparisons offered as needed to VSIPL++, so that both are covered by the concepts presented.
- Definition of a handful of minimal(istic) thread-safe programming models is made, ones that emphasize optimistic locking, and demonstrations of performance impact on e600 and Core i7 platforms. This mode of operation does not introduce the need for new APIs into the standards. These basic models and their logical extensions inform of a new taxonomy of VSIPL/VSIPL++ programming models.
- A discussion of minimal concurrency management within VSIPL/VSIPL++ implementations, and OS/runtime implications of the duties of the library, vs. the underlying OS (*e.g.*, Linux vs. VxWorks).
- Discussion of thread affinity implications are also covered in terms of user program goals, operating system behavior, and realistic limits/potential for continued program portability.
- Definition of potential high performance extensions to the minimal models that rely strictly on optimistic locking, and how to reduce pessimistic locking; mentioning the kinds of locking appropriate to different situations is also covered (*e.g.*, short vs. long-term locking);
- Discussions of what more generalized programming models would be that involve several degrees of inter-thread sharing, and/or internal library concurrency. For instance, specific thread-aware code, and thread-optimized behavior can be supported better by extending the APIs, and by using the object-based (resp, object-oriented) natures of VSIPL and VSIPL++ to designate the kind of sharing of specific objects. For instance, some objects may not be shared or be single executed, and they need not be subject to pessimistic locking; others may need to be shared in ways that force the library itself to lock.
- Implications of multithreaded programs running on systems with memory hierarchy, heterogeneity, and segmentation are mentioned as well. In particular, implications of the admit-release paradigm in the face of multi-level concurrency in the user program and the implementations, as well as heterogeneous memory are discussed.
- Further implications on VSIPL/VSIPL++ programming when multi-kernel GPU scheduling occurs in tandem with multicore

processors running single- or multi-threaded VSIPL, as is now possible with CUDA+Fermi GPUs from NVIDIA. We offer specific possible solutions for how to manage such cases, and indicate remaining open issues.

- Performance overheads between single-threaded and multi-threaded libraries will be demonstrated with the VSI/Pro(R) family of VSIPL implementations. We find that these can be quite low, particularly if the underlying operating system already provided thread-safe memory management, but less so on systems where inter-task (inter-process) locking is relatively expensive. Compiler issues will be touched on briefly.
- A review of the implications in the performance-portability-productivity space of these new hardware architectures and programming models, and specifically to emerging "threats" to portability at acceptably high performance.
- Extra implications because of C++ in the APIs, in the user programs, and in the implications are discussed briefly.
- Finally, a call to action for prompt and on-going standardization is made.

Relevance to the HPEC Audience

Current defense-oriented programs (meaning the organizations adopting the standard and the software artifacts alike) of the VSIPL standards have access to multicore processors routinely at this time: Intel Core i7, Atom, and Freescale 8641D processors, in particular, with others to follow. This militates that a clear resolution on how to use VSIPL in performance-portable modes without undue overheads be defined and agreed on by the community. Furthermore, it suggests that high performance implementations will need to offer "production multithreaded" and "production single threaded" libraries as well as runtime switches or added API, in order to avoid accidental overheads for programs that continue to use single threaded programs. In addition, it should be noted that VxWorks SMP mode, which presents its task-oriented execution model, inherently requires thread-safe VSIPL implementations even when independent user programs are running. The on-going adoption of this operating system is another reason why this paper is extremely timely and crucial to the continued viability of VSIPL and VSIPL++.

The impact of high performance, flexible, and "profile"-oriented solutions to the concomitant opportunities and challenges of fine-grain concurrency will ensure that the VSIPL and VSIPL++ standards continue to be broadly usable in the context of fine-grain parallelism in multicore and SMP systems both in embedded and cluster/cloud environments. Conversely, lack of immediate attention to these concerns will clearly limit the usability of these standards moving forward, or will limit portability as different implementers provide alternative strategies for multithreaded and/or multikernel environments. This paper's presentation will ideally raise awareness to the importance of defense program deciders about the need for careful consideration on how to adopt fine grain concurrent systems and how to use VSIPL and VSIPL++ effectively in such systems, including adapting legacy applications for such systems (*e.g.*, refreshes, code reuse on new defense programs) in ways that exploit the fine-grain concurrency effectively, quickly, correctly, and robustly.

References

1. VSIPL Standards, URL: www.vsipl.org ; accessed May 19, 2010.
2. "Using Concurrency, Chapter 3", http://gcc.gnu.org/onlinedocs/libstdc++/manual/using_concurrency.html, accessed May, 19, 2010.
3. Intel Core i7 Processor: http://www.intel.com/products/processor/corei7/index.htm , accessed May 19, 2010.
4. e600 series SMP Freescale Processors: http://www.freescale.com/webapp/sps/site/overview.jsp?code=DRPPCDUALCORE , accessed May 19, 2010
5. e500 series SMP Freescale Processor: http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3bTdG25E4&tid=SAC&gclid=CLq6xrXs36ECFdtB5godh2wVKA , accessed May 19, 2010.
6. Cain, Kenneth, and Sroka, Brian, "Experiences in Porting an Existing Application to the VSIP API," MITRE Corporation, July 22, 1997, presented at the VSIP Meeting, URL: http://www.vsipl.org/VSIP_Exp.pdf , accessed May 19, 2010.
7. Exascale Software Study, Kogge et al, URL: http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf , accessed May 19, 2010.