Array Building Blocks: A dynamic compiler for data-parallel heterogeneous systems

Chris J. Newburn (CJ)

Summary

Hardware platforms are getting harder to program effectively, as they grow in complexity to include multiple cores, SIMD hardware, accelerators (e.g. GPUs and Intel's Knights Ferry - formerly Larrabee) and even clustering support. The challenge is to provide an efficient means of exploiting parallel performance to the masses of novice programmers. At least two things are required to solve this problem: 1) a language interface that enables ease of use without the perils of parallel programming, and 2) a compiler infrastructure that takes a high-level specification of what to do, and portably maps it onto a variety of heterogeneous hardware targets.

This tutorial uses Intel's newly-released Array Building Blocks (ArBB) to illustrate the challenges and potential solutions in this space. ArBB is a dynamic compiler infrastructure that can compile or JIT for both SIMD and thread parallelism on symmetric multi-processor, distributed, and accelerator targets. Parallelism is exposed to this infrastructure via an embedded language, e.g. a library-based C++ API, using aggregate data types and operators. It enables safety and debuggability by construction. It allows applications with kernels that are recoded with minimized effort to be compiled once using standard compilers, and then be dynamically retargeted to platforms that haven't even been invented yet by simply switching runtime libraries. We put ArBB in the context with a variety of other programming models, including CUDA and OpenCL.

Target Audience

This tutorial may be of practical interest to language experts who are interested in parallel language design, to educators who may want to leverage this material to teach parallel programming models for emerging architectures, and to practitioners who may benefit from our experience implementing compilers to address customer-driven concerns. The presentation of new language features, application examples, optimization techniques that enable efficient offload to accelerators, and the demonstration of speedup and debugging support on both standard laptops and pre-production acceleration hardware are some of the aspects of this tutorial that could draw participation.

Outline

Overview

The objectives of this tutorial are several:

- Highlight the need for new data-parallel APIs that offer features not currently available through current offerings or their extensions.
- Highlight the need for a new generation of retargetable heterogeneous compilers.
- Describe a specific compiler, based on ArBB, that meets those requirements, and that is a major part of Intel's parallel, heterogeneous and accelerator platform enabling strategy.
- Explore in detail ArBB's compiler architecture, optimizations, phases, code generation strategies, and interactions with threading and heterogeneous runtimes.
- Provide the audience with both an intuitive overview and substantive training in a product that's been publicly released as a beta.

- Show the relevance of this programming model to several application domains, and describe how it relates to and interacts with other programming models.
- Demonstrate the product on both laptops and moderately parallel systems with Knights Ferry (formerly Larrabee) compute accelerators.

Motivation

Parallel programming and debugging is hard. The majority of today's programmers prefer to think serially, and want to avoid the productivity pitfalls of data races and debugging of concurrent code. Those programmers that do pursue the last bit of performance by blocking for caches and targeting implementation-specific features in their code often end up with code that's very difficult to port and maintain. In contrast, ArBB provides a higher-level abstraction through which programmers can expose data parallelism with a serial sequence of operators on aggregate data types. Only performance-sensitive kernels need to be modified, and features of the ArBB API, such as elemental functions, try to minimize the amount of code restructuring. ArBB's abstraction allows programmers to focus on expressing the "what", and leaves the "how" to the compiler infrastructure, by default. If expert programmers want exert more control, the hooks are there to do that.

There's an increase in thread and SIMD parallelism in today's platforms. ArBB offers a means of being able to seamlessly harvest both of these, without having to mix programming models. But it does interoperate with other programming models, such as Threading Building Blocks (TBB), that enable use of thread parallelism. And there's an increasing pace in architectural changes that are visible to software, such as the memory abstraction model in heterogeneous or distributed systems, the advent of new instruction sets, and increasing sizes and layers of cache hierarchy.

Access to parallelizing compiler infrastructure should not be limited by the front end. Modern languages provide productivity improvements, e.g. through garbage collection and scripting. ArBB provides a virtual machine interface through which additional language front ends may be added, beyond the initial C++ offering.

Details

I'll describe how ArBB works, using diagrams, application examples, and demonstrations. An application will by modified so that its kernels use ArBB, then the code will be compiled and debugged with a standard compiler in a stock integrated development environment (IDE). An overview of ArBB's API will follow. Aggregate types and operators will be explained, and I'll explain how data movement and synchronization works across disjoint data spaces.

We'll take a look under the hood at ArBB's compiler architecture. We'll examine its phases, optimizations, and code generation strategies. We'll explore how dynamic compilation invoked, what code generation strategies are used, and how retargeting and cross-platform support works. And I'll provide some details on the threading and heterogeneous runtimes work.

ArBB is moving to provide an open standard for its virtual machine, allowing other frontends to be added, other than C++. We expect to illustrate this with Python, and perhaps other examples. ArBB's first-class representations of code objects (closures) enable explicit manipulation of the code generation process. These apply to compiled objects at both the IR stage and final code generation phase.

One section of the tutorial, that reflects a lot of thought and discussion within Intel's tool groups, covers programming models. I'll compare and contrast different programming models along several axes and show how they can be layered. I'll cover common issues, like elemental functions, array notation, and perhaps the specification of ordering and locality.

I'd like to wrap up with results of some performance analysis and a demonstration of ArBB's applicability across several different application domains. I expect to demonstrate performance on both a laptop and (if logistics allow) pre-production Intel accelerator hardware, called Knights Ferry, formerly known as Larrabee.

References

- Chris J. Newburn, Michael McCool, Byoungro So, Zhenying Liu, Anwar Ghuloum, Stefanus Du Toit, Zhi Gang Wang, Zhao Hui Du, Yongjian Chen, Gansha Wu, Peng Guo, Zhanglin Liu, Dan D. Zhang, *Intel® Array Building Blocks: A Retargetable, Dynamic Compiler and Embedded Language*, In Proceedings of Code Generation and Optimization, 2011
- Anwar Ghuloum, Eric Sprangle, Jesse Fang, Gansha Wu, and Xin Zhou. *Ct: A Flexible Parallel Programming Model for Tera-scale Architectures.* Technical Report White Paper, Intel Corporation, 2007
- Michael McCool, *Data-Parallel Programming on the Cell BE and the GPU Using the RapidMind Development Platform*, GSPx Multicore Applications Conference, 2006.

Bio

Chris (CJ) Newburn serves as a feature architect for Intel's Intel64 platforms, and has contributed to a combination of hardware and software technologies that span heterogeneous compiler optimizations, middleware, JVM/JIT/GC optimization, acceleration hardware, ISA changes, microcode and microarchitecture over the last thirteen years. Performance analysis and tuning have figured prominently in the development and production readiness work that he's done. At present, his primary responsibilities include serving as an architect of Array Building Blocks, and representing software tools' interests in our many-core accelerator efforts. He likes to work on projects that span the hardware-software boundary, that span organizations, and that foster collaboration across organizations. He has submitted nearly twenty patents and has numerous journal and conference publications. He helped start CGO, has served on several program committees, as a journal editor, and as an NSF panelist. He wrote a binary-optimizing, multi-grained parallelizing compiler as part of his Ph.D. at Carnegie Mellon University. Before grad school, in the 80s, he did stints in a couple of start-ups, working on a voice recognizer and a VLIW mini-super computer. He's glad to be working on volume products that his Mom uses.