EMBRACING HETEROGENEITY - PARALLEL PROGRAMMING FOR CHANGING HARDWARE WITH THE MERGE FRAMEWORK

Michael D. Linderman, Teresa H. Meng
Dept. of Electrical Engineering, Stanford University

mlinderm@stanford.edu

Introduction
The computational requirements of informatics applications are growing at an exponential rate.

Goal: Enable ‘productivity’ programmers to exploit state-of-the-art heterogeneous multicore computers

Heterogeneous systems, which integrate specialized accelerators such as GPUs or FPGAs alongside multicore general-purpose processors (GPPs), can deliver the scalable performance needed by demanding informatics applications.

Programming such systems is a keen challenge, however. New tools are needed. Unfortunately there is no ‘one’ compiler that can meet all our needs. The solution lies in integrating many narrowly focused processor-application-domain, and vendor-specific toolchains. The Merge framework does exactly this.

Merge abstracts away the complexity of heterogeneous systems behind a backward-compatible function metaprogramming interface. Applications, written by domain experts, target these processor-agnostic interfaces, which are automatically mapped to compute-expert-created processor-specific modules.

Composition & Coordination
The heart of any integration technique is a common abstraction. Traditional compiler-driven approaches, built around a fixed set of primitives, e.g. short vectors, cannot abstract all the computational features we are interested in. A different, more inclusive abstraction is needed.

Instead of a particular arithmetic or memory operation, Merge is built around functions, a type of inclusive primitive available in most programming languages, and a common “hub & spoke” system organization.

Merge has three key components:

1) Function Intrinsics: Encapsulate specialized code in legacy compatible C/C++ function wrappers
2) Bundling: Function intrinsics for the same computation are collected into “bundles” that provide indication and introspection to dispatch system automatically maps applications to particular variants based on programmer-supplied annotations
3) Parallel Frameworks: High-level parallel programming languages built on the encapsulation and bundling capabilities.

The dispatch meta-wrappers provide a level of indirection between applications and specific function intrinsics that facilitates the automatic selection of implementations for particular input and machine configurations. The ordering represents the automatic selection of implementations for particular input and machine configurations. The ordering represents the Merge compiler’s inference of a “good” mapping.

The meta-wrappers support two usage modes:

1) Direct Usage: Directly call the simple wrapper show above. Completely hides any heterogeneity from the end-user. Intended for programmers who want the simplest, most backward-compatible programming model.
2) Indirect Usage: Other wrappers enable introspection into the available variants and their dispatch annotations. Programmers can write their own domain-specific schedulers.

Parallel Frameworks
High-level parallel programming languages help raise the level of abstraction for both expert and non-expert programmers. The Merge prototype includes a parallel language built around map-reduce semantics.

The map-reduce language is encapsulated inside C/C++ functions, and implements a specialized scheduler that attempts to improve performance by dynamically distributing work across different processors in the system organization.

Merge Framework

Application
Parallel Framework
Legacy Libraries
C/C++ Library Interface
Merge Components
Library Analyzer & Bundler
Library Descriptions
Parallel System
Compiler & Optimization
Library Meta-wrappers

Merge software stack: Merge components highlighted in blue. Each layer builds on the previous, and all present interfaces compatible with legacy SW infrastructure.

Further Reading

Acknowledgements
This work is funded in part by the FCBP Focus Center Circuit and Systems Solutions (C525), under contract 2003-CT-888, and the Intel Corporation.

Application Example
Real-time Spike Sorting for Neural Prostheses
Neural prosthetics systems restore lost communication and movement functionality for patients with spinal cord injuries or neural degenerative diseases.

The prosthetic is controlled by neural signals recorded with electrodes implanted in the cortex. As the number of electrodes grows, real-time spike sorting, which disambiguates among different neurons recorded on the same channel, is becoming increasingly computationally demanding.

Each stage implements well-understood kernels that can be parallelized by computing experts within and across channels.

Function Variants
1. Single channel IIR filter for single processor
2. Multi-channel IIR filter for single processor with SSE
1. Single channel FFT for single processor
2. Multi-channel FFT for single processor with SSE
1. Single channel k-means for single processor
2. Multi-channel k-means for GPU

Results
We implemented a set of kernels on both heterogeneous and homogeneous test platforms. Significant speedups relative to the single-core reference implementation are achieved on both platforms using the same source program.

Speedup for Core 2 Duo CPU and X3000 Integrated GPU

Speedup for Core 2 Duo CPU and X3000 Integrated GPU

Speedup on 32-way homogenous SMP

Further Reading