Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs

1. Abstract

Task-based parallel programming frameworks offer compelling productivity and performance benefits for modern chip multi-processors (CMPs). At the same time, CMPs also provide packed SIMD units to exploit fine-grain data parallelism. Two fundamental challenges make using packed SIMD units with task-parallel programs particularly difficult: (1) the intra-core parallel abstraction gap; and (2) inefficient execution of irregular tasks. To address these challenges, we propose augmenting CMPs with intra-core loop-task accelerators (LTAs). We introduce a lightweight task in the instruction set to efficiently execute loop-task execution and an LTA microarchitectural template that can be configured at design time for different amounts of spatial and temporal task pipelining. The LTAs can efficiently execute both regular and irregular loop tasks. Compared to an in-order CMP baseline, CMP+LTA results in an average speedup of 4.5. (9.9 area-normalized) and similar energy efficiency. Compared to an out-of-order CMP baseline, CMP+LTA results in an average speedup of 2.2. (1.3 area-normalized) and also improves energy efficiency by 3.2. Our work suggests augmenting CMPs with lightweight LTAs can improve performance and energy efficiency in both regular and irregular loop-task parallel programs with minimal software changes.

2. Motivation

Loop-task parallelism is a common pattern usually captured with the parallel for primitive, where a loop task function is applied to a block of data. There are two fundamental challenges that make using packed SIMD units in the loop-task context particularly difficult:

- **Intra-Core Parallel Abstraction Gap.** Two fundamentally different parallel abstractions reduce productivity: tasks for inter-core parallelism (e.g., TBB) and packed SIMD for intra-core parallelism (e.g., AVX). Auto-vectorization and explicit vectorization are challenging to perform since tasks can be arbitrarily complex and software-only abstractions are not known at compile time, potentially preventing "multiplicative speedup."

- **Inefficient Execution of Irregular Tasks.** Loop tasks are often complex with nested loops and function calls, loop invariant computation, loop variables, array accesses, and atomic operations compared to the scalar implementation. Converting branches into arithmetic results in wasted work, extra memory alignment and/or data transformations adds overhead, scatte/gather accesses often have much lower throughput, and there's an inefficient algorithmic approach may be required for vectorization. All of these reasons derive from the fact that the microarchitecture for packed SIMD extensions is fundamentally designed to excel at executing regular data parallelism as opposed to the more general loop-task parallelism.

3. LTA Software

**Programming Interface**

```c
void avxdd(int dest[], int src0[], int src1[], int size)
{
    for (i = 0; i < size; ++i)
        dest[i] = src0[i] + src1[i];
}
```

We propose a new jalr.lta instruction that has the same semantics as normal irregular function call jalr, but serves as a hint to the underlying hardware that the function has the special signature of loop-task.

**LTA-Enabled Work-Stealing Runtime**

The LTA-enabled work-stealing runtime still recursively partitions loop tasks into subtasks to facilitate load balancing until the range is less than the core task size, but then uses the jalr instruction. If an LTA is available, the GPP can potentially use the LTA to further partition the core task into sub-tasks, each responsible for a smaller range of iterations. The LTA groups subtasks into task groups which execute a set of instructions in lockstep (i.e., same instruction), exploiting structure for efficient execution.

4. LTA Hardware

**Spatial and Temporal Task Coupling**

Task Grouping

5. Cycle-Level Evaluation

**Performance vs. HW Resource**

This work was supported in part by NSF CAREER Award #144044, NSF XPS Award #1377430, NSF CNS Award #1523737, NSF IIS Award #1737054, AFOSR YIP #FA9550-15-1-0194, and donations from Intel, NVIDIA, and Synopsys. The authors acknowledge and thank Scott McKeown and Amin Waje for their early work on the LTA HPC modeling, Jason Setter and Wei Gang for their work with scalable processor PyMTL modeling, and David Solder for access to the Intel Atom Phi 5110G.

---

**Contact Author:** Shuming Jiang, 471 Rhodes Hall, Ithaca, NY 14853, sj13@cornell.edu


**School of Electrical and Computer Engineering, Cornell University**