Coping with very High Latencies in Petaflop Computer Systems

Sean Ryan, José N. Amaral, Guang Gao, Zachary Ruiz, Andres Marquez, Kevin Theobald*

Computer Architecture and Parallel Systems Laboratory, University of Delaware, Newark, DE, USA. http://www.capsl.udel.edu

The very long and highly variable latencies in the deep memory hierarchy of a petaflop-scale architecture design, such as the Hybrid Technology Multi-Threaded Architecture (HTMT) [13], present a new challenge to its programming and execution model. A solution to coping with such high and variable latencies is to directly and explicitly expose the different memory regions of the machine to the program execution model, allowing better management of communication. In this paper we describe the novel percolation model that lies at the heart of the HTMT program execution model [13]. The Percolation Model combines multithreading with dynamic prefetching of coarse-grain contexts. In the past, prefetching techniques have concentrated on moving blocks of data within the memory hierarchy. Instead of only moving contiguous blocks of data, the thread percolation approach manages contexts that include data, program instructions, and control states.

The main contributions of this paper include the specification of the HTMT runtime execution model based on the concept of percolation, and a discussion of the role of the compiler in a machine that exposes the memory hierarchy to the programming model.

1 Introduction

The Hybrid Technology Multi-Threaded (HTMT) Architecture project [15] has the goal of designing a petaflop scale computer by the year 2007. Such a machine will use a number of unconventional technologies such as: processors and inter-connection networks built from super-conducting processing elements (called SPELLs [32]), networks based on RSFQ (Rapid Single Flux Quantum) logic devices [11], “Processor In Memory” (PIM) technology [20], high-performance optical packet switched network technology [7], optical holographic storage technology [26], and fine grain multi-threaded computing technology [16].

In this paper we introduce a new program execution model developed for the future HTMT machine. An important characteristic of the HTMT machine is the availability of a large number of very high performance super-conductor processing elements (SPELLs) with a modest amount of single-flux-quantum cryo-memory (CRAM) that can be accessed with a relatively low latency [32]. The latency for the next levels in the memory hierarchy (e.g. SRAM and DRAM) will be several orders of magnitude higher than a CPU cycle time in the SPELLs.

Our analysis shows that hiding the latencies of the deep memory hierarchy in the HTMT architecture is a great challenge; existing multi-threaded execution/programming models may not be able to cope with such latencies, where

* emails: {ryan, amaral, ggao, ruiz, marquez, theobald}@capsl.udel.edu
even a small percentage of cache misses can have disastrous effects upon performance. As a result, we introduce a new program execution and programming model, the *percolation model*, in order to meet this challenge.

Percolation can be considered to be a combination of multi-threading with dynamic prefetching and reorganization of the data as well as the threads which use the data into coarse-grain contexts. Prefetching in the past has concentrated on moving blocks of *data* within the memory hierarchy. Instead, functions, data, and control states are combined and sent in a single *parcel*. A parcel cannot be percolated to CRAM until all functions and data associated with it are available for transport. The latencies incurred in gathering data for the parcel and its component threads will then be made to overlap. Also, under this model, any data destined to be reused by the same thread is guaranteed to be already stored locally.

The programming model for percolation makes use of program directives to specify what pieces of data will be needed by a portion of the code and how the data should be organized into parcels before any code is actually sent to the high speed processors. Processors-in-memory (PIMs) provide the necessary capability to perform such data transformations and to prepare parcels. These parcels, once completed, are percolated to the fast processing units. Any results produced by the computation are percolated back to the memory after the computation in the processing units is complete. These results might then undergo (reverse) data transformations by the PIMs.

The unique memory model of the HTMT and its ramifications are discussed in section 2. Next, in section 3 we introduce, at a conceptual level, the percolation model of program execution as a means of coping with the architectural constraints discussed above. We then divide the conceptual diagram into phases of execution and introduce the runtime system (RTS) that implements the percolation model (section 4). This leads into a discussion of the role of a compiler for a petaflops machine (section 5) and of the next steps in evaluating the percolation model for the HTMT (section 7).

## 2 Memory Model

Conventional architectures present to the programmer the appearance of a uniform address space below the registers, with caching and paging hiding the real details of the hierarchy from the programmer. This luxury is not available in the HTMT model. Instead, each level of memory is considered to be a "buffer" of the next level, e.g., super-conducting memory is a buffer of the SRAM, which itself is a buffer of DRAM, and so on. Unlike in traditional cache organization, these buffers are directly addressable. Memory allocation and data movement at each level may be explicitly controlled if necessary [13].

We assume that the entire HTMT memory address space is explicitly divided into regions: a CRAM region, an SRAM region, a DRAM region, and an Optical

---

1 Remote transactions such as data movement are handled asynchronously. Sync slots can be used to signal their completion.