Summary of Parallel Computation

We have learned a substantial amount about parallel computation. Today, we summarize the main conclusions.

1

But, first latency tolerance

2

Latency -- Just Do Something Else

In theory memory a delay of λ is not a show stopper; simply switch to other work while waiting for a memory value to be returned

Requires (in theory) P log P threads, but it's actually λP, and grows however λ grows

Threads are often abundant, but it is difficult always to have λP threads, e.g. at decision points

3

Hardware Implementations

- The idea of switching to execute instructions from another thread when memory latency stalls processor has been around a long time
  - Early Honeywell machines used related ideas
  - Denelcor HEP (~1982) designed by Burton Smith had 8 threads "in the air" at once
  - The 1990s saw several efforts to build such machines
- We will study two designs
  - Alewife from Anant Agarwal's group at MIT
  - The Tera Computer from Burton Smith

4

The Hardware Solutions

- Focus on keeping 1 processor busy in the presence of long latencies to shared memory, but expect to use many such processors
  - Use multithreading
  - Requires no special software as long as the compiler can produce more threads than processors
  - Handles both predictable and unpredictable situations
  - Handles long latencies even as they grow
  - Doesn't affect the memory consistency model, i.e. shared variables must be locked or use other mechanism

$$\text{utilization} = \frac{\text{work\_time}}{\text{work\_time+switching+idle}}$$

5

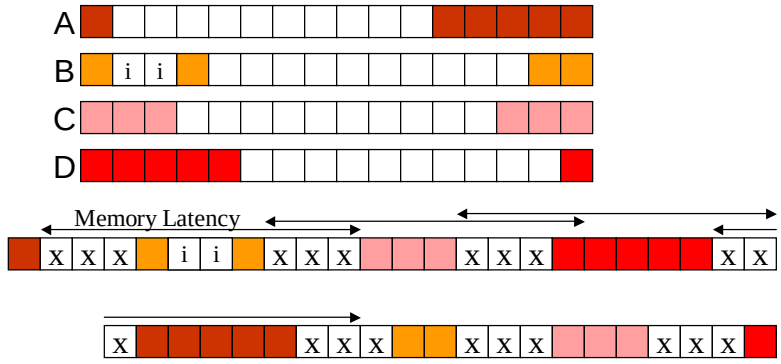
Two Techniques for Multi-threading

- Blocked multithreading [Alewife] is like timesharing … continue to execute until the thread is blocked, then switch
  - Has lower hardware impact
  - Good single thread performance
- Interleaved multithreading [Tera] switches execution on threads on each cycle
  - Lower logical switching penalty
  - Greater impact on hardware design
- Keeping multiple contexts is essential

6

Four Threads Using Blocked Approach

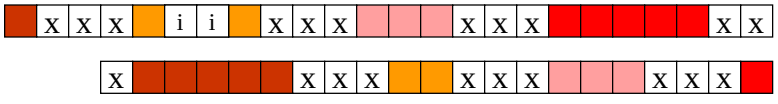
- Threads make memory reference every few instructions



7

Utilization of Blocked Approach

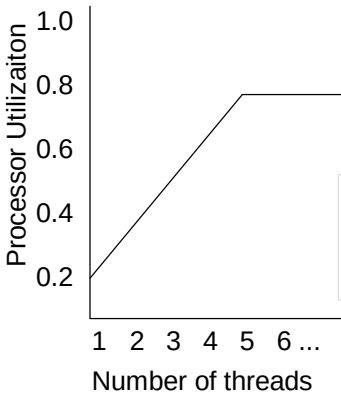
- Total instruction times: 45
- Total work instructions: 24
- Total switch time: 21
- Total wait time: 0
- Utilization = 24/45 = 53%



8

Benefits of Available Threads

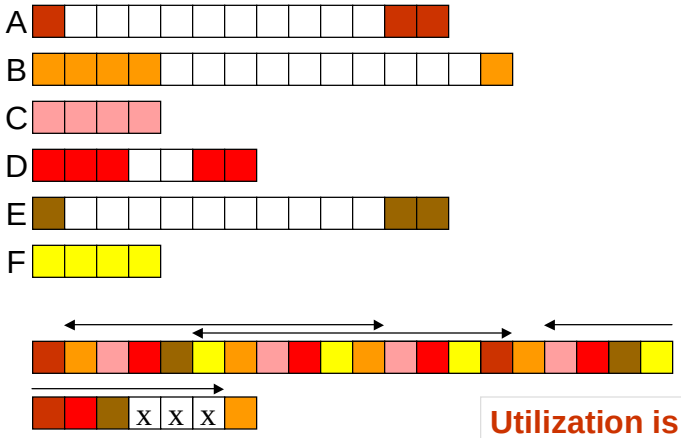
- For the blocked approach the availability of ready threads improves utilization



Switching time is overhead that is not recovered

9

Six Threads With Interleaved Approach

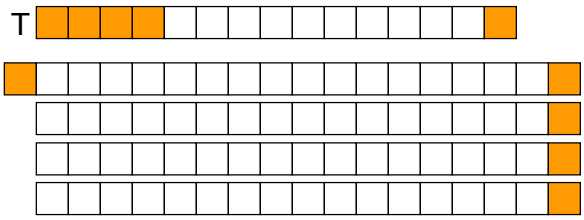


Utilization is 24/27 or 89%

10

Basics of Tera Design

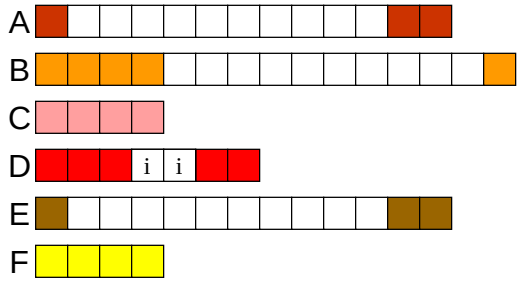
- Instructions are [arithmetic, control, memory] or [arithmetic, arithmetic, memory]
- Ready instructions issue on each tick, but there is a 16 tick minimum issue delay for consecutive instructions from a thread



Converts latency from 15 ticks to 69 … U = 7%

11

Six Threads Revisited (Tera Design)



Utilization is 23/70 or 33%

12

More On Tera II

- Since there is a 16 instruction minimum issue delay, it takes 16 threads to execute sequentially *without* latency hiding
- Each (memory) instruction has a 3 bit tag telling how many instructions forward are independent of this memory reference (in this thread)
- Average memory latency without contention is 70 cycles

13

More On Tera III

- Each processor has 128 full contexts
- Synchronization latency can even be covered
- When everything works, the Tera should approximate a PRAM

Think of the Bulk Synchronous model with completely decentralized supersteps

14

Summary

Our Original Goal

Goal: To give a good idea of parallelism

- Concepts -- look at problems with “parallel eyes”
- Algorithms -- different resources to work with
- Programming -- describe the computation without saying it sequentially
- Languages -- reduce control flow; increase independence
- Architecture -- HW support to share memory not?
- Hardware -- the challenge is communication, not instruction execution

Start with HW, and review (pop) to Concepts ...

Routing

- Chaos routing is effective because “non-minimal” adaptivity can by-pass congestion
  - Light traffic, randomize routes over a regular, symmetric, consistent networks, avoids creating hot spots; no point where packets can get “stuck”
  - Moderate traffic, wait in a node for a route to clear, this is better than “hot potato” which *must* route
  - Heavy traffic and faults, deroute in the wrong direction to move around the problem

Higher throughput, lower latency, higher load-carrying capacity than other routers

17

Chaos Routing (continued)

- Deadlock is not possible because of packet exchange protocol
- Probabilistically livelock-free,
  - As good or better than deterministically livelock-free in practice
  - Solves difficult (but rare) problem for adaptive routers by randomizing, and gambling
- Chaos is not perfect; not good with wormhole
  - Inefficient for long messages; use two nets or pick a variable length packet with large-ish maximum

18

Networks

- Full cross-bar is not practical
- Direct and Indirect Networks are alternatives
  - Indirect, e.g. Ω-network
    - Has only "long" paths of O(log P), no nearest neighbor
    - Multiple references to a location can collide, so try combining at the switches
    - In fact, exploit combining with Fetch&Add -- it's better for shared memory than Test&Set because it "schedules"
    - Both Fetch&Adds and Load/Stores can be combined
    - Combining requires "smart" switches that slow net
    - Analysis shows combining opportunities are rare; hot spots due to colliding references to different locations is the problem

F&A + combine is smart but flawed

Networks (continued)

- Direct Networks
  - Short, nearest neighbor paths are available
  - Adaptive routing techniques are available
  - Much more asynchronous; NIC is extra processor
  - Different load properties for different architectures
    - Non-shared memory, network carries little overhead
    - Shared memory, network carries coherency protocol messages, which can be "proportional to the sharing"
  - n-ary, d-cubes are realistic topologies
    - Torus is better because of symmetry
    - Fat trees also work; hypercube has "log P node degree"

Direct (regular) networks are only choice

Architecture

Main architecture decision: hardware support for shared memory or not?

- Non-shared memory architectures are successful
  - Simpler designs, means faster designs
  - Leave memory management to software/programmer
  - A single address space is easy and useful
  - "Proper HW support for shared memory" is still unknown and getting less realistic as technology improves
  - Avoid message passing and its copy/marshal overhead
  - One-sided communication (shmem) is very efficient because it reduces communication's synchronization
  - Shmem allows "strided communication" with pipelining

Single address space, 1-sided communication is best

Architecture (continued)

- Shared memory
  - Technically very difficult -- and therefore slow -- to keep memory coherent
  - DSM can be implemented by a directory scheme
    - Record sharers/dirty features for each cache line
    - Directory can double the memory requirements of machines, though some simplifications are possible
    - Follow distributed version of coherency protocol because no bus for defining "timing sequence"; use mem location
    - Homework showed: Even limited sharing can cause much traffic
    - Invalidations, acknowledgements increase with sharing

DSM best when not used, i.e. manage mem yourself

Architecture (continued)

- Symmetric-multiprocessors (SMPs) are an effective way to share memory on a small scale
  - Cache controllers snoop memory bus
  - The bus becomes the "time sequencing" point of the system, where modification order is defined
  - Various protocols speed performance with greater complexity
  - "DSM Homework sharing" would be reasonably efficient on SMP … cost only about '2x' over non-share
  - Bus is *serially* used, limiting generalization to small #s

SMP is a standard architecture

23

Languages

- Shared memory is difficult to use (races, synch); not efficiently implemented; not realistic
- Message passing with sequential language (C | Fortran) + (MPI | PVM) is current standard
  - Least common denominator -- runs everywhere
  - A huge amount of work (6x code explosion)
  - Only the API is standard; semantics vary, making any program implementation-specific; limit porta'ty
  - Message passing is costly on architectures with "good" communication, e.g. 1-sided, SMP

Use msg passing if you must; but there's a better way

Languages (continued)

- ZPL 1st (and still only) parallel language with performance model (WYSIWYG)
  - Designed from first principles to help programmers
  - No explicit concurrency, communication, or synch.
  - Programmer is insulated from details, but it is possible to write efficient solutions with WYSIWYG
  - Compiler is heavily optimized, both seq. and para.
  - The communication abstraction is Ironman -- four procedures that mark sender's/receiver's active regions -- uses native communication of machine

ZPL is convenient and efficient

25

Languages (continued)

- ZPL's performance model
  - Allows programmers "to keep their distance" from the implementing hardware -- portable!
  - Relies on abstract machine -- CTA
    - CTA gives key structural information, memory reference time, processors, characteristics of interconnection net
    - CTA gives parallel costs; vN defines sequential costs
    - Give ZPL's runtime model, work & data allocation
    - Describe costs of ZPL's constructs in CTA terms
  - No absolute performance possible, but relative is good enough for quality programming -- performs!

The most significant idea of this class

26

Programming

Everyone thinks shared memory is the natural parallel extension of sequential computing: "Ignore memory reference time like vN model, and let HW give the flat memory illusion"

- Memory reference time is key to good algs:
  - Find maximum is the example
    - Best ignore-memory-time (PRAM) is Valiant O(log log P)
    - Best consider-memory-time (CTA) is tournament O(log P)
    - (Actual?) implementation of Valiant's alg O(log P loglog P)
    - Actual implementation of tournament O(log P)

PRAM hides a critical cost => it's hard to get results

27

Programming (continued)

- The CTA replaces the PRAM as a realistic, but still abstract model of parallel computation
  - CTA models all existing hardware, but is "far enough away" to be independent of all
  - CTA is concerned with a few features, processors, non-local memory reference time, λ, interconnect, which has unspecified topology, low degree
  - Practicing programmers writing message passing code are in effect using the CTA
  - CTA is key to expressing costs of HLL like vN

A machine model separates SW & HW development

28

Concepts

- The powerful parallel computation ideas are:
  - Pipelining, perform some operations and then pass the task along for completion by other units
  - Overlap, perform communication & computation simultaneously since they need separate resources
  - Partition, form independent (as possible) tasks and assign separate processors to each
  - Most parallel algorithms use a combination of these
    - Languages should support these concepts
    - ZPL does overlap and partitioning for all computations up to available resources, and has abstraction for pipelining

More abstractly: Decompose into independent parts

Concepts (continued)

- Matrix multiplication -- the most studied parallel algorithm
  - Many solutions; van de Geijn,Watts SUMMA best
    - Uses broadcast communication of rows/columns
    - Restructures the problem to “use data completely”
    - Efficiently uses temporary space
    - Most natural and convenient (and efficient) ZPL program
    - Other algs show ‘problem space promotion’ technique

Problem space promotion is a parallel programming technique in which a problem with d dimensional data d is *logically* solved in a higher dimension, usually d+1

Avoid iteration

30

Concepts (continued)

- Summary for successful parallel computation
  - Rather than using a shared memory abstraction, use the CTA model; it reflects costs accurately
  - Use ZPL for programming to get convenience, speed and portability; use MP as last resort
  - Be suspicious of claims like the “problems” with shared memory have been solved by new machine
  - When choosing architecture, prefer support for global addressing, 1-sided communication, point-to-point network, (randomizing) non-minimal adaptive routing, SMP nodes

The perfect parallel machine has yet to be built

31

Summary’s Summary

- This has been an enjoyable class to teach.
- Good luck with the remainder of your MS degree.

32