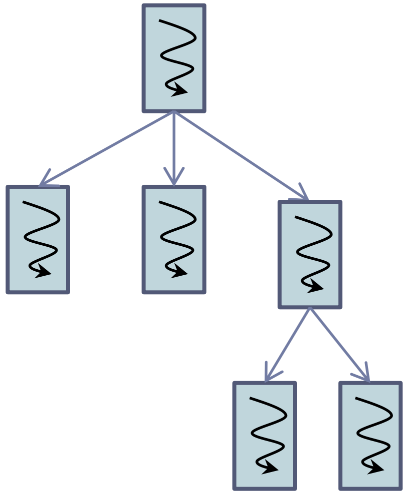
MassiveThreads: A Thread Library for Massively Parallel Machines

Kenjiro Taura @ University of Tokyo
(Joint work with Jun Nakashima, Shigeki Akiyama, Sho Nakatani)

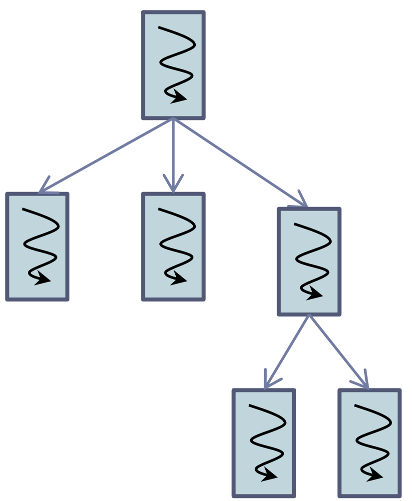
Motivation and Long Term Goals

- Machines are getting larger, more hierarchical, and more complex
- Reconcile performance and programmability based on task parallelism + global address space in post-peta scale machines



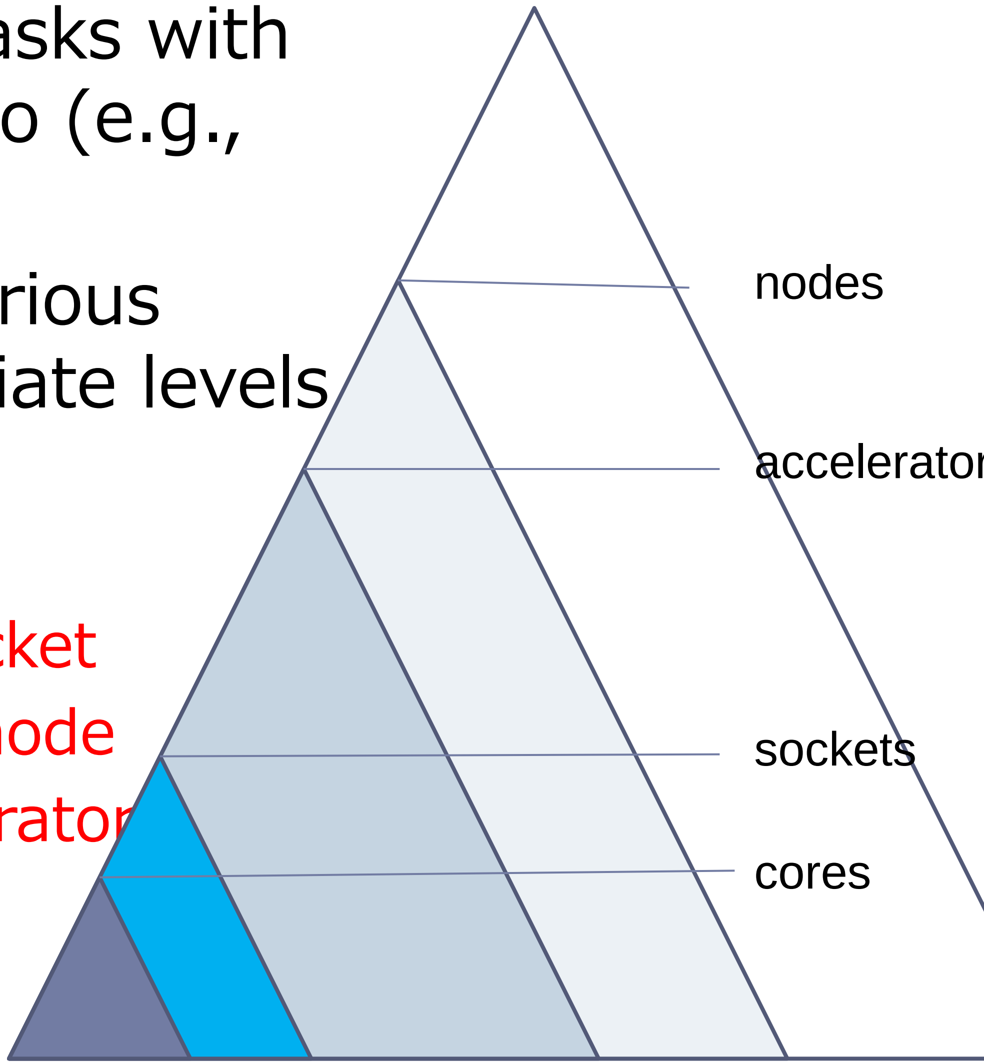
What are Task Parallelism, and What is It Good for?

- It allows programs to launch a new task at any point in execution (cf. SPMD)
- It encompasses:
  - Parallel loops (arbitrarily nested)
  - Parallel recursions (divide-and-conquer)
  - Latency hiding



What is It Good for?

- dividing tasks into subtasks with good compute/data ratio (e.g., ORB, SFC)
- mapping subtasks of various granularities to appropriate levels of machines
  - Across nodes
  - Across cores within a socket
  - Across sockets within a node
  - Between CPU and accelerator



Project Roadmap

- Design philosophy
  - Reusable & language neutral (compiler independent)

Domain specific languages (join work ongoing with Maruyama et al.)

A minimalist extension to C++

Plugging into other languagess
(Chapel)

Distributed memory multithreading (task parallel) library
MassiveThreads/DM

Efficient multhreading (task parallel) library
MassiveThreads

Flexible PGAS library
DMI

Today's Talk

- MassiveThreads
  - A library that brings efficient task parallelism into YOUR language

Domain specific languages (join work ongoing with Maruyama et al.)

A minimalist extension to C++

Plugging into other languagess (Chapel)

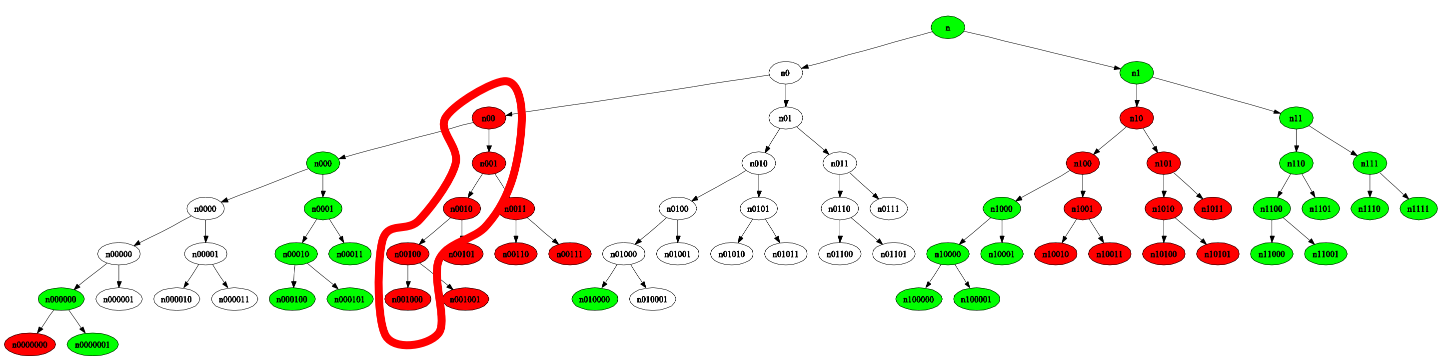
Distributed memory multithreading (task parallel) library MassiveThreads/DM

Efficient multhreading (task parallel) library MassiveThreads

Flexible PGAS library DMI

Implementing Task Parallelism

- The basic strategy is well-known at least on shared memory [Mohr ‘91 et al.]
  - Each worker maintains deque
  - Work-first and LIFO execution by each worker
  - FIFO stealing: an idle worker steals the oldest continuation in another worker’s deque
- A variety of implementation along this idea



Various Strategies

- Frontend (code generator) managing frames yourself
  - Original Lazy Task Creation, Concert, Cilk
- Java bytecode rewriting
  - Satin (Java)
- Binary (assembly) rewriting
  - StackThreads/MP

Need cogen to make "continuation" stealable
Not easy for YOUR language to take advantage

- Task parallel libraries
  - TBB, Java fork-join

LIFO but not work-first

- Thread libraries
  - Qthreads, Nanos, MassiveThreads

MassiveThreads Objective

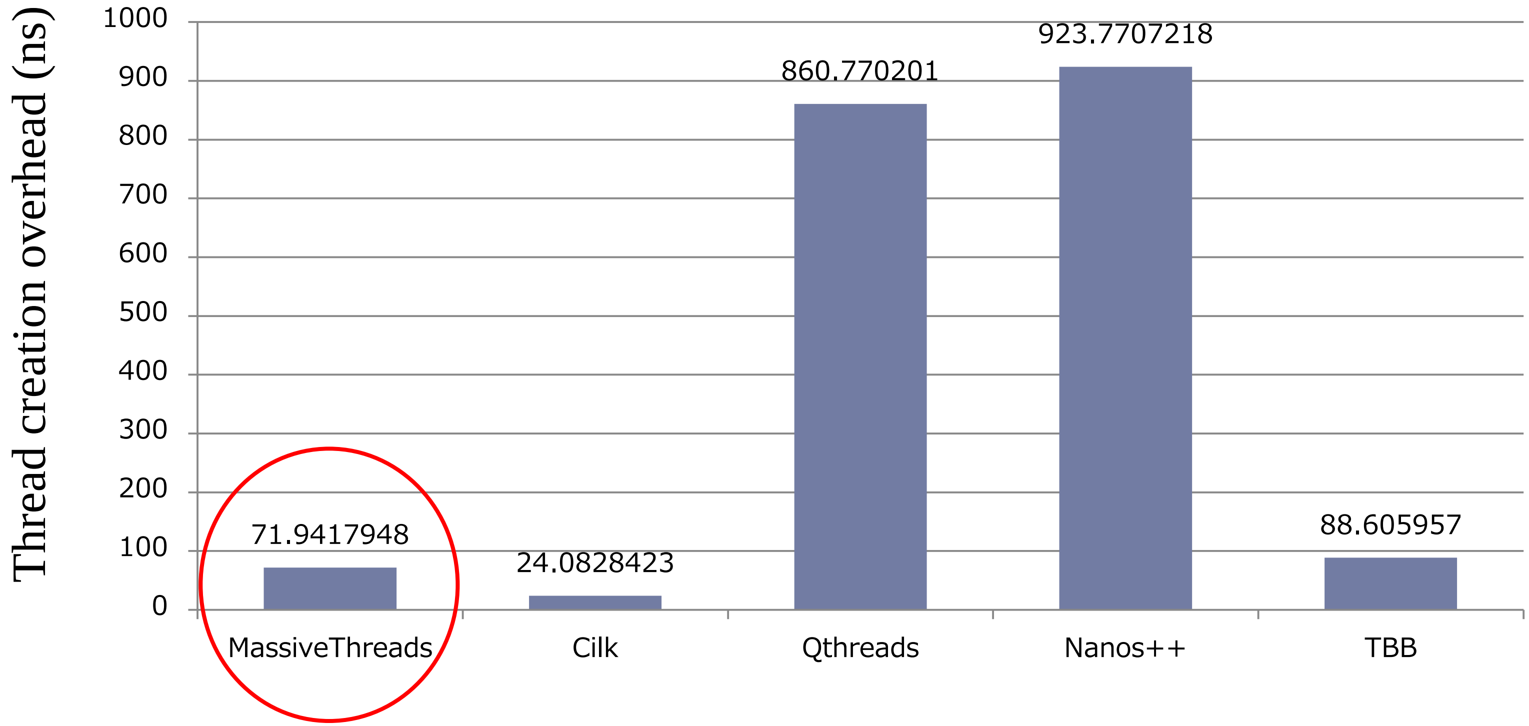
- Provide an efficient implementation of task parallelism on MPPs that:
  - is straightforward to use and take advantage
    - Want a task? Link this one and create a thread!
  - gives language developers a good compilation target
    - C/C++ with native C compilers or anything that can link our library
    - surface language is completely your choice

MassiveThreads Specific Features

- Pthreads-compatible API
- lightweight threads
  - just call “ptheads_create” and it is as cheap as “tasks”
- works with native C/C++ compilers (just as Pthreads does)
  - making your language -> C/C++ translation straightforward
- Pthreads-like I/O semantics
  - Blocking I/Os switches to another user-level thread without losing a parallelism
  - Important for distributed memory machines

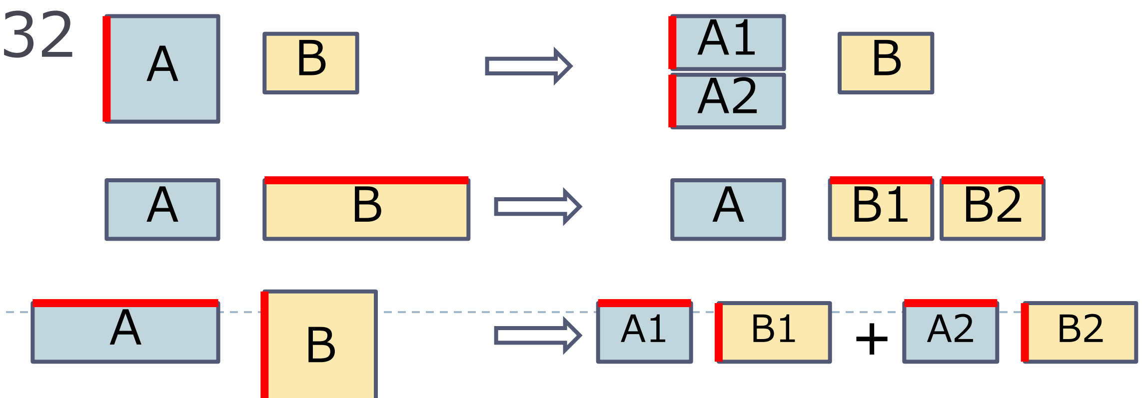
Microbenchmark

- thread creation & destruction : 72ns (150 cycles)



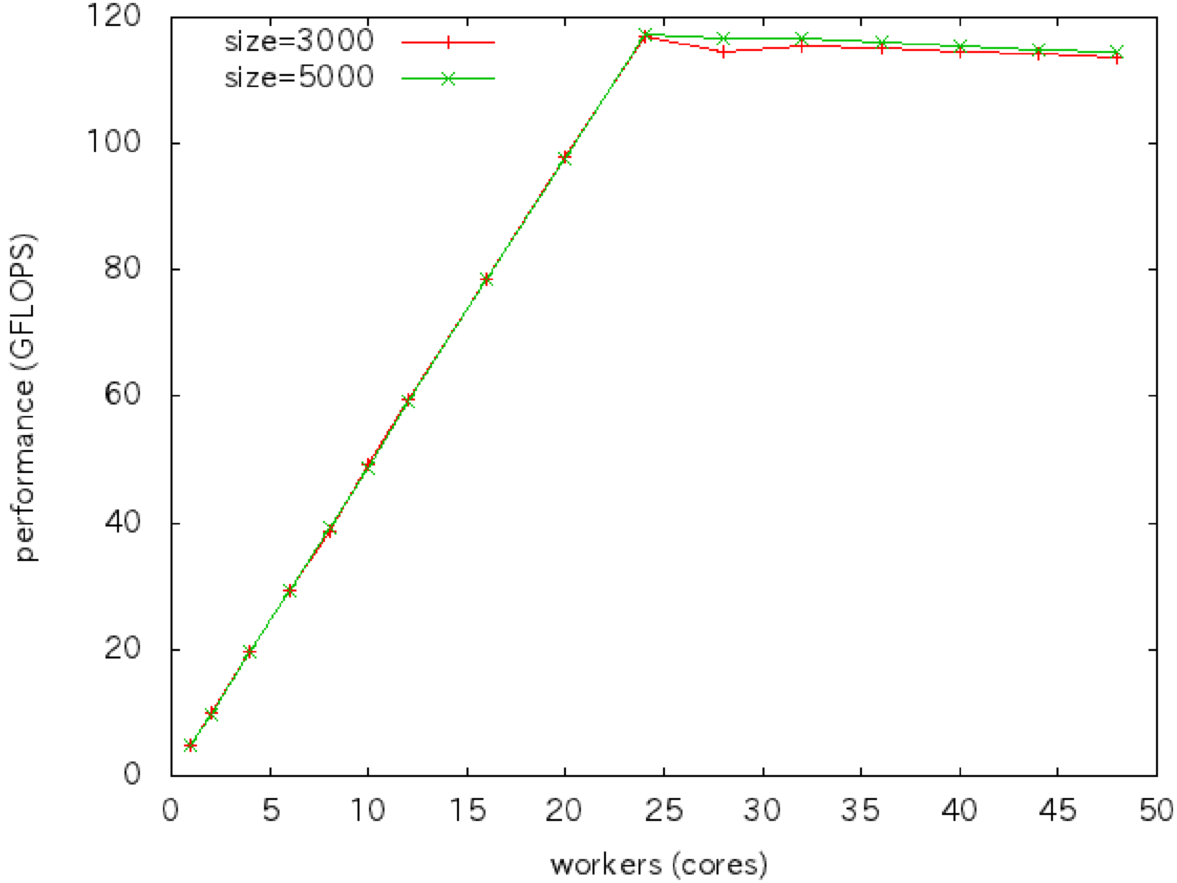
Demo: Plugging MassiveThreads into Your Program without Recompilation

- Platform
  - Nehalem 2.0GHz 24 cores, 48 hardware threads
  - Intel Threading Building Block-like task parallel (task_group class) built on top of Pthreads API (50 lines)
  - Run the same program with
    - Pthreads
    - MassiveThreads
- Program：recursive matrix multiply
  - Divide until size ≤ 32



Speedup

- Almost ideal speedup up to physical cores (24)
- Absolute performance (1/3 of the theoretical peak) is due to the leaf routine only modestly optimized

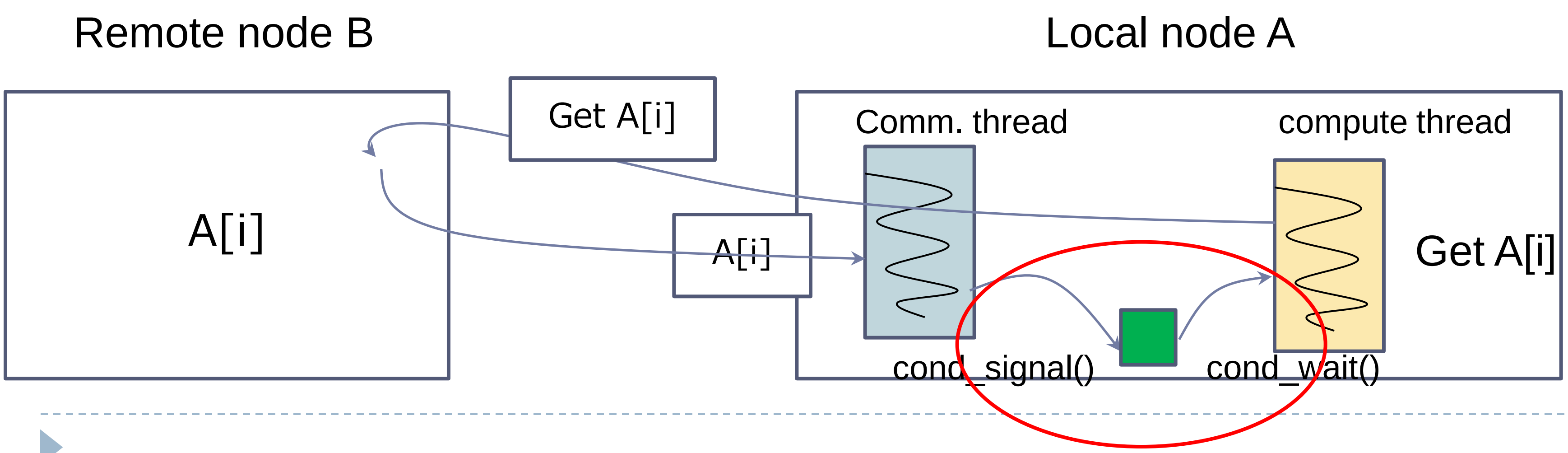


MassiveThreads : Pthreads-like Blocking I/O Semantics

- When a thread blocks on I/O, it switches to another user-level thread
- Many user-level threads block the underlying worker thread, losing a degree of parallelism
- Obviously useful for multithreaded server apps
- Important for parallel languages on massively parallel machines too

Why is Pthreads-like I/O Semantics Important?

- Runtime systems for distributed memory machines often use a dedicated thread for inter-node communication ("comm. thread")
- Problem : how to get the "comm. thread" and compute threads synchronized?



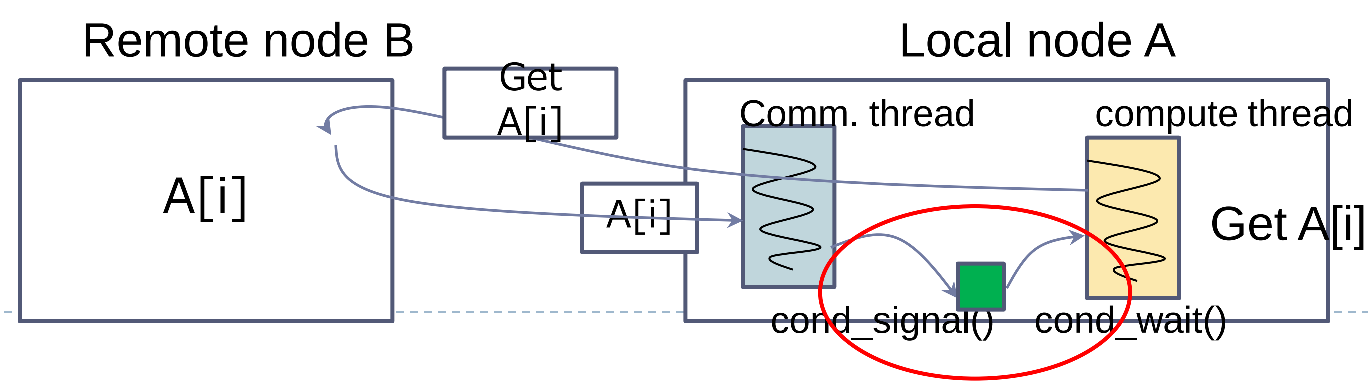
How to Synchronize with the Comm. Thread?

- With typical user level thread packages, we need to decide whether the comm thread should be a Pthread or a user-level thread
  - user-level thread ⇒ the comm thread can't issue blocking I/O
  - Pthread ⇒ synchronization must use a Pthread

when

With MassiveThreads, it becomes trivial

Use Pthread for eveything



A Few Words about Implementation

- Fast multithreading
  - Worker-local free list for fast stack allocations
  - Inlined assembly context switching
  - Separate stack and thread control block for prompt reuse of stack space
- Blocking I/O
  - Intercept relevant system calls (send, recv, select, read, write, etc.)
  - Blocking I/Os are issued as non-blocking I/Os

Ongoing Work

- Plugging MassiveThreads into Chapel
  - Working to make it available as part of next Chapel release
- More "careful" work stealing
- A global address space library for irregular data structures
  - Caches
  - Again, the goal is language neutrality
- MassiveThreads/DM
  - Native threads migrating across nodes

Thank you!

- MassiveThreads availabel from Google code at:
  - http://code.google.com/p/massivethreads/