Towards a codelet-based runtime for exascale computing

Chris Lauderdale
ET International, Inc.

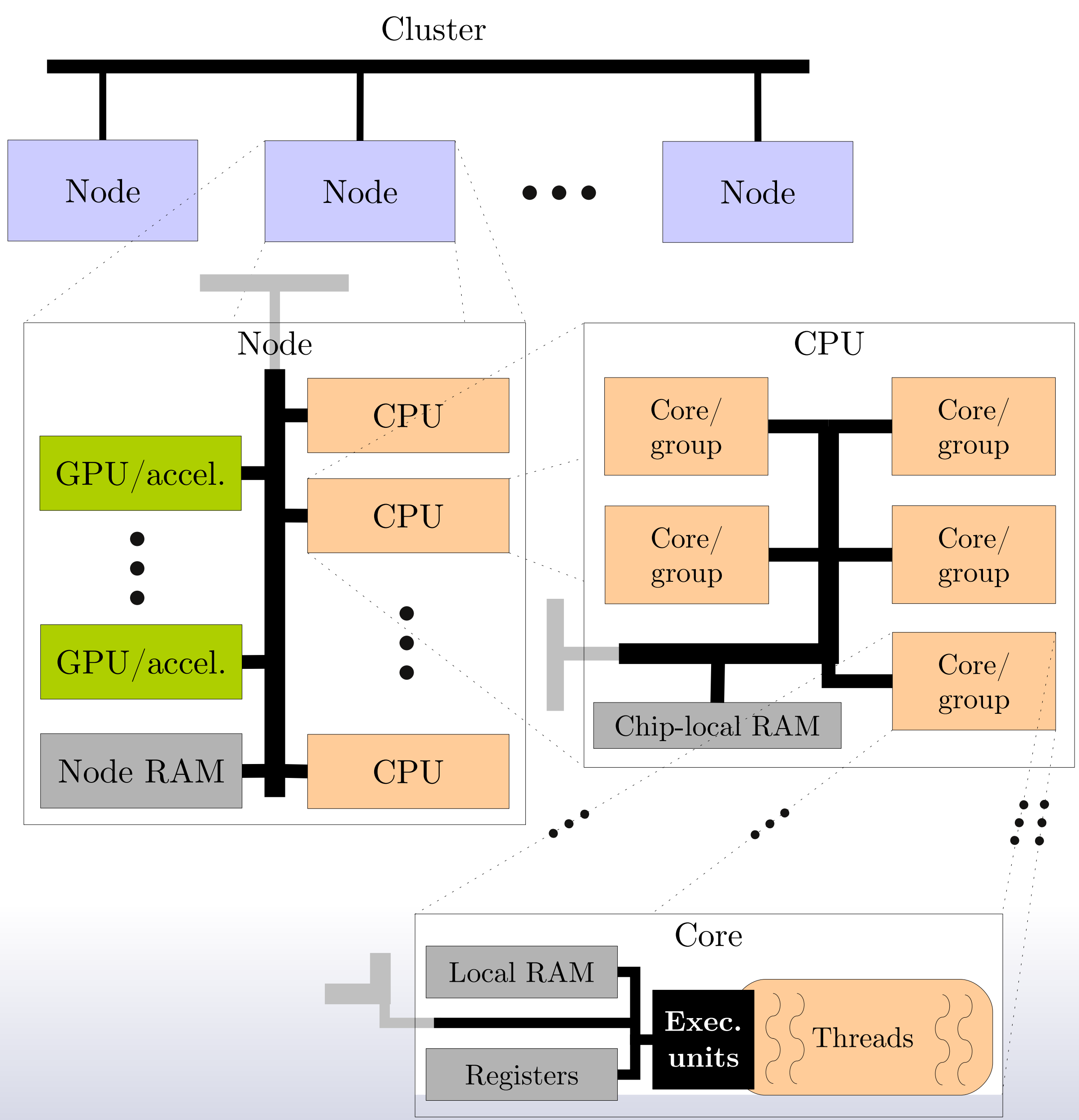
What will be covered

- Problems & motivation
- Codelet runtime overview
- Codelets & complexes
- Dealing with locality & heterogeneity
- Related work & conclusion

Introduction

- Can't reach exascale by continuing past trends
- Need something to
  - Expose and coordinate parallelism
  - Control data and execution locality
  - Abstract platform- and location-specific details
  - Unify software interface for supercomputing

Abstract machine model



Hardware-related problems: Scalability

- Present & future reliance on thread-level parallelism for performance increases
  - Can't keep increasing clock rate
  - Can't keep relying on instruction-level parallelism
- Memory access
  - More cores $\rightarrow$ higher access latency, power cost
  - Not practical to use coherent caches
  - Small core-/chip-local memories simplify hardware but complicate software
  - Need a way to hide access latencies and cross address spaces

Hardware-related problems: Heterogeneity

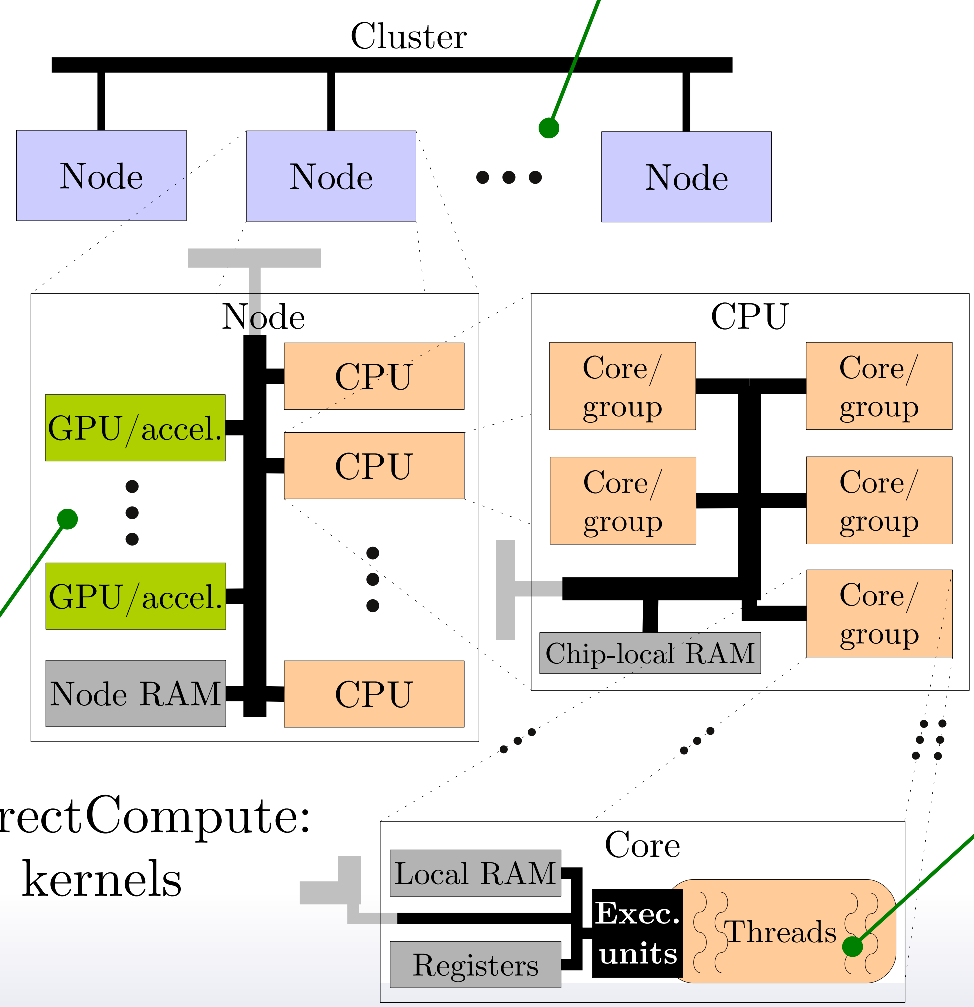
- Increasingly common
- Good solution for
  - Effective utilization of space/power on chip
  - Accelerating matrix-/vector-related operations
- Difficult to actually use in software
  - Special APIs for accelerators
  - Must statically partition work or duplicate code
- Need to handle more transparently (unify and coordinate support software)

Software-related problems

- Reliance on sequential processing, coherent memory
- Can use multithreading for parallelism, but
  - High address space/memory overhead for stack
  - High overhead to create, manage, switch threads
  - Stack must remain in fixed address range for its lifetime
- Need a way to sidestep blocking, expose fine-grained parallelism

Existing software frameworks

MPI, SHMEM: Explicit data transfer



OpenCL, CUDA, DirectCompute:
Self-contained SIMD kernels

OpenMP: Parallel for-all
Cilk: Parallel recursion

Existing software frameworks

- MPI, SHMEM
  - Must explicitly transfer data to/from specific nodes
  - Are not thread-safe in general (specific to implementation)
- OpenMP, Cilk, TBB
  - OpenMP & Cilk geared to specific algorithm types
  - TBB is C++-only; Cilk is C-only, but techniques could be applied to C++/FORTRAN
  - Work only in one address space
  - Uniform, coherent memory assumed
- OpenCL, CUDA, DirectCompute
  - OpenCL and Direct3D device contexts not thread-safe; CUDA is
  - Must explicitly coordinate CPU and GPU
- Existing frameworks achieve specific goals, but do not interact well.

Codelet runtime overview

Present software stack:

Application
System libraries
Operating system
Hardware

Proposed software stack:

Application
Codelet runtime
System libraries
Operating system
Hardware

Present execution model:

Function calls
[User-mode threads]
OS/HW threads

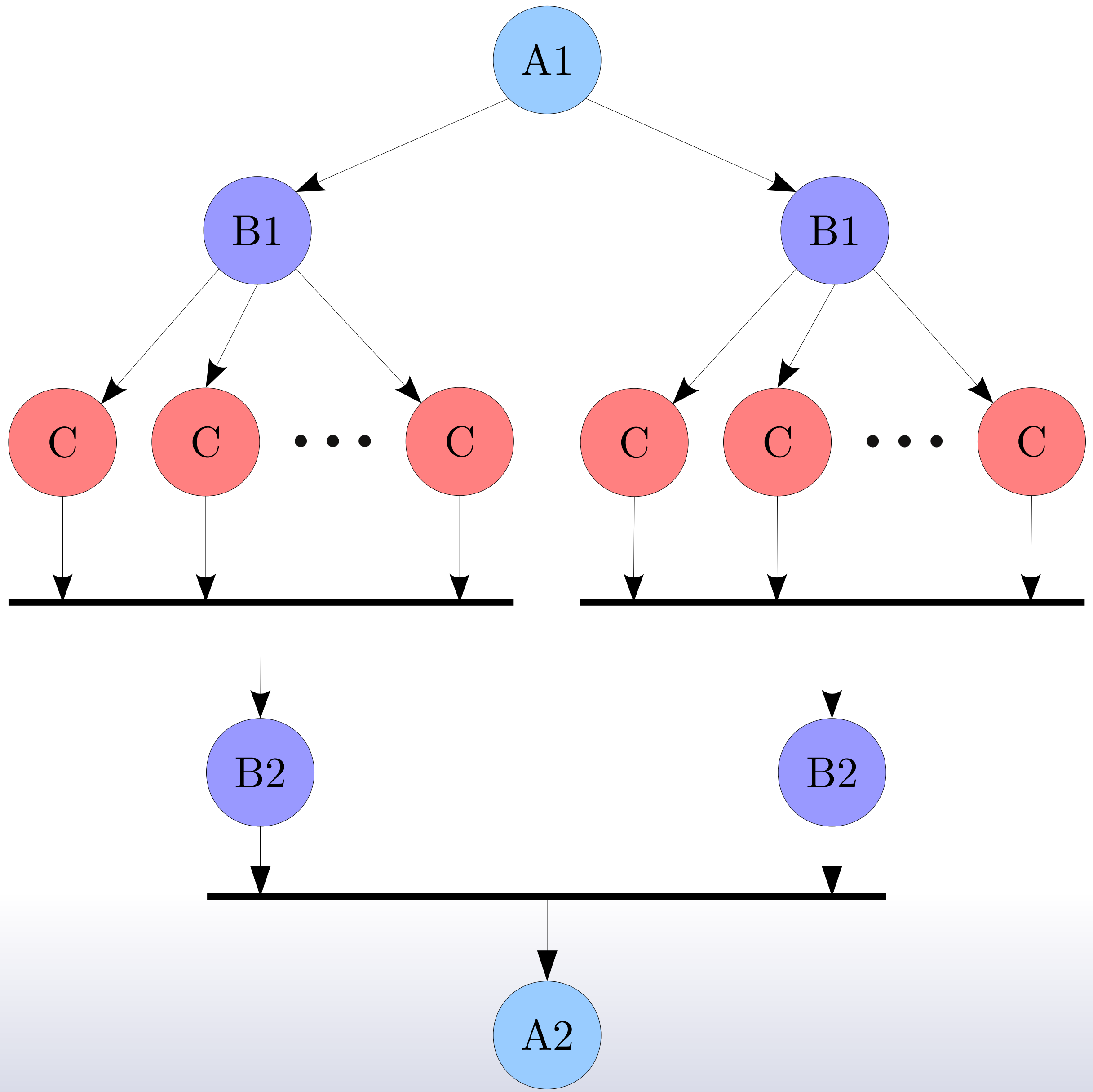
Proposed execution model:

[Function calls]
Codelet dispatch
[User-mode threads]
OS/HW threads

Codelet runtime overview: Codelets

- Break application into smaller pieces (=**codelets**)
  - Codelets shouldn't block or run indefinitely
  - Must explicitly spill/fill at codelet boundaries
- Low-overhead hiding of long-latency operations
  - One codelet starts an operation, another catches the result
  - Runtime provides for inter-address-space mobility
- Simple & rapid exposure of fine-grained parallelism
  - Makes scalability easy—just provide work and something will run it

Example: Dual parallel for-all loops



Codelet runtime overview: Locales

- High-level description of available hardware
  - Region-bound processing+storage capabilities: **locale**
  - Exposed API for placing codelet execution & data
- Codelets+locales enable transparent handling of heterogeneity

Codelets

- Fundamental unit of scheduling/execution
- Represented by in-memory descriptor
- **Run fork**: Work to be performed to advance program state.
- **Cancel fork**: Work to be performed to back out program state, in case an error is encountered.

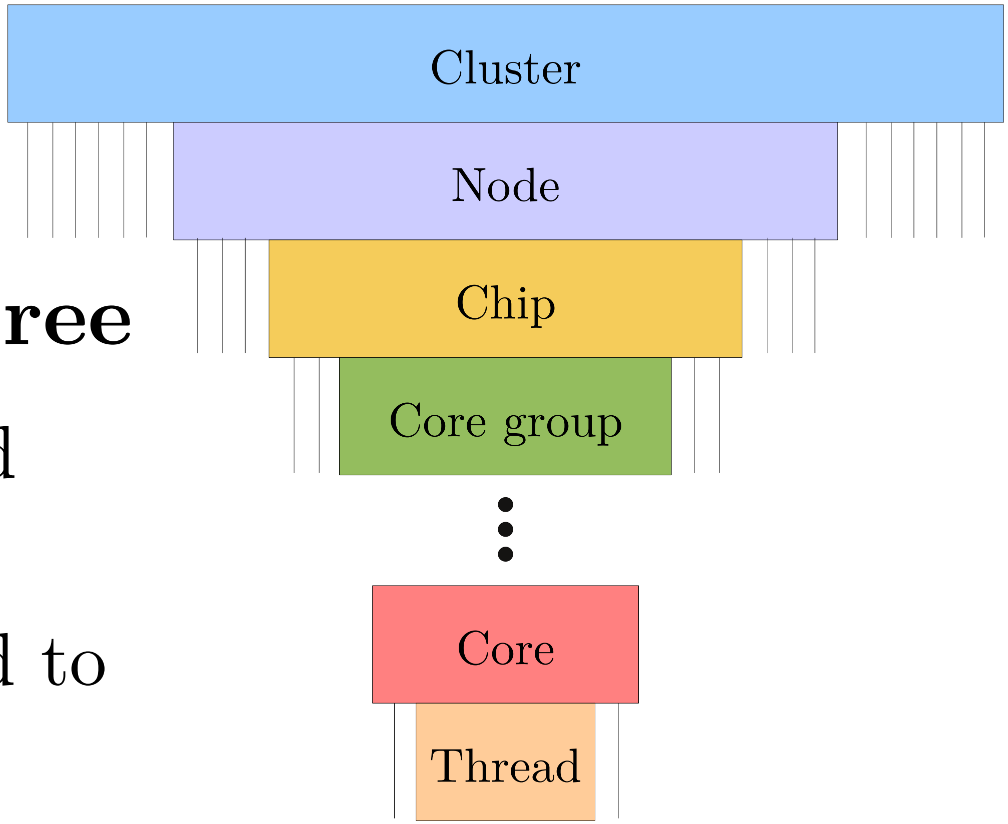
Codelet complexes

- **Codelet complex**: Ad-hoc group of ≥1 codelet(s) that cooperate to complete some task.
- Can specify **chain codelet** & context when starting
- Complex must **chain**—run or cancel its chain codelet—before completing. Used for:
    - Input cleanup
    - Passing return values, taking further inputs
    - Catching and resuming from errors

Codelet/function interoperability

- Codelets/complexes used as implementation of HLL functions:
  - Chain codelet+context corresponds to return IP+SP
  - Input to chain corresponds to return value
  - Error to canceled chain corresponds to thrown exception
- Functions used in implementation of codelets:
  - Run/cancel forks implemented as functions
  - Runtime calls fork function to dispatch codelet
  - Return from fork function = end of codelet
- Complexes can be wrapped as functions and vice versa

Locality awareness



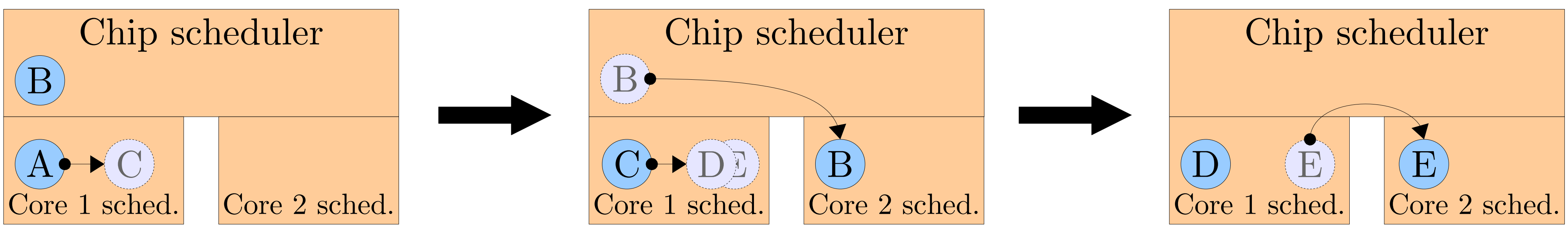
- System components grouped into a **locale tree**
  - Each locale has attached scheduler & allocator
  - **Leaf locales** correspond to threads
  - Higher-level locales manage children's resources collectively
  - Schedulers/allocators push and pull work around the hierarchy

Handling heterogeneity

- Global locale tree shared throughout runtime
- Locales describe associated hardware details
- Code format/ISA differences
  - Codelets are identified globally, but different descriptor data may be used in different locales
  - Can provide different run/cancel forks for different architectures using same descriptor

Scheduling and allocation

- Leaf schedulers/allocators manage time/space on a particular thread, higher-level can delegate
- Application can specify sooner/later ordering



B has been scheduled to chip.
A is running on core 1.
Core 2 is idle.
A schedules **C** to core 1.

B taken by core 2.
C runs on core 1.
C schedules **D** & **E** to core 1.

D runs on core 1.
E stolen by core 2.

Applicability to algorithm classes

- Fork-join-style algorithms
  - Recursion-based
    - Can parallelize multiway-recursive algorithms
    - Application-specified scheduling order limits parallelism blowup
  - Data-parallel/SIMD
    - Can do parallel for-all over locales to distribute work
    - Work stealing automatically balances load afterwards
- Dataflow algorithms
  - Can register codelet instances to catch data availability
  - Can use locale-based routing to walk around graphs

Related work

- Basis for codelets: Gao et al.'s theoretical model
  - Dropped theoretical limitations
  - Added cancellation and chaining semantics
- Locales closely related to Habanero hierarchical place trees
- Existing frameworks:
  - MPI, SHMEM, OpenMP, Cilk, TBB, OpenCL, CUDA, DirectCompute (already addressed)
  - ParalleX (model) and HPX (runtime implementation)
    - Many higher-level constructs
    - Can implement PX constructs on top of a codelet runtime

Ongoing/future work

- SWift Adaptive Runtime Machine
  - Version 0: Experimental prototype; available for download
    - Reduced scheduling capability, codelet semantics, allocator support
  - Version 1: Under development

Conclusion

- Need a new execution model for exascale
- Codelet runtime model enables
  - Scalability
    - Feed codelets to the runtime, don't rely on threading
    - Unified model for entire cluster
  - Portability
    - Single portable runtime interface
    - Platform differences can be dealt with by runtime
  - Better hardware utilization
    - Automatic load balancing
    - Transparent use of heterogeneous components

Questions/comments?

SWARM v0 download: `http://etinternational.com/swarm`