CRAY
THE SUPERCOMPUTER COMPANY

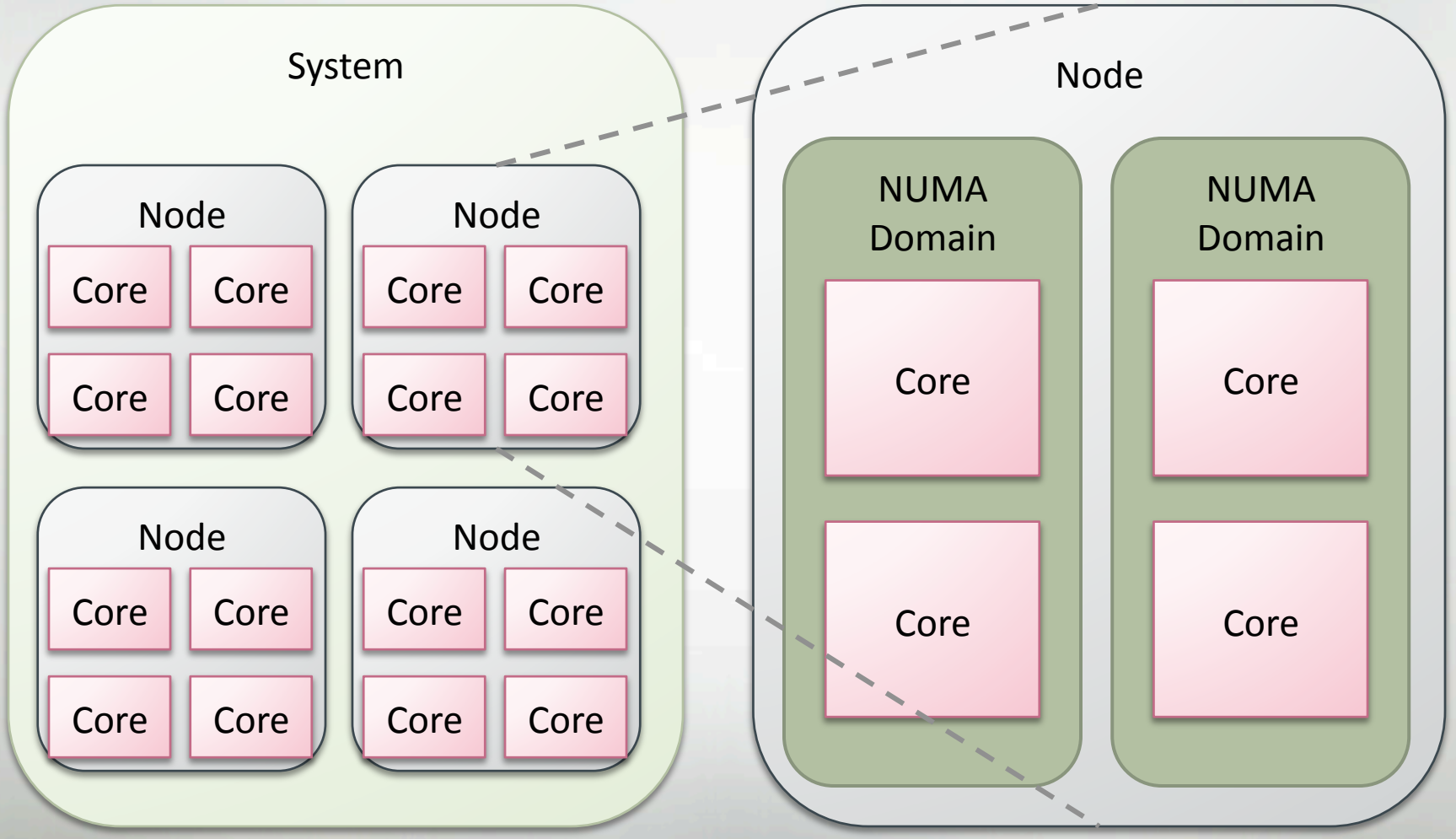
Hierarchical Locales: Using Sublocales to Boost Performance on NUMA nodes

Casey Battaglino, Georgia Tech

Tom Hildebrandt, Cray Inc.

Background

- "Traditional" Architectures
- Recent Architectures



Presently, Chapel's "locale" model reflects the "traditional" architecture.

Hierarchical Locales Interface

```
// Hierarchical locales interfaces added:
class locale {
  // Architectural description
  proc addChild(child : locale) : void;
  proc getChild(child_id : int) : locale;
  iter getChildren() : locale;
  // Process control
  proc initTask() : void;
  proc endTask() : void;
  // Memory management
  proc alloc(nbytes : int) : object;
  proc free(x : object);
}
```

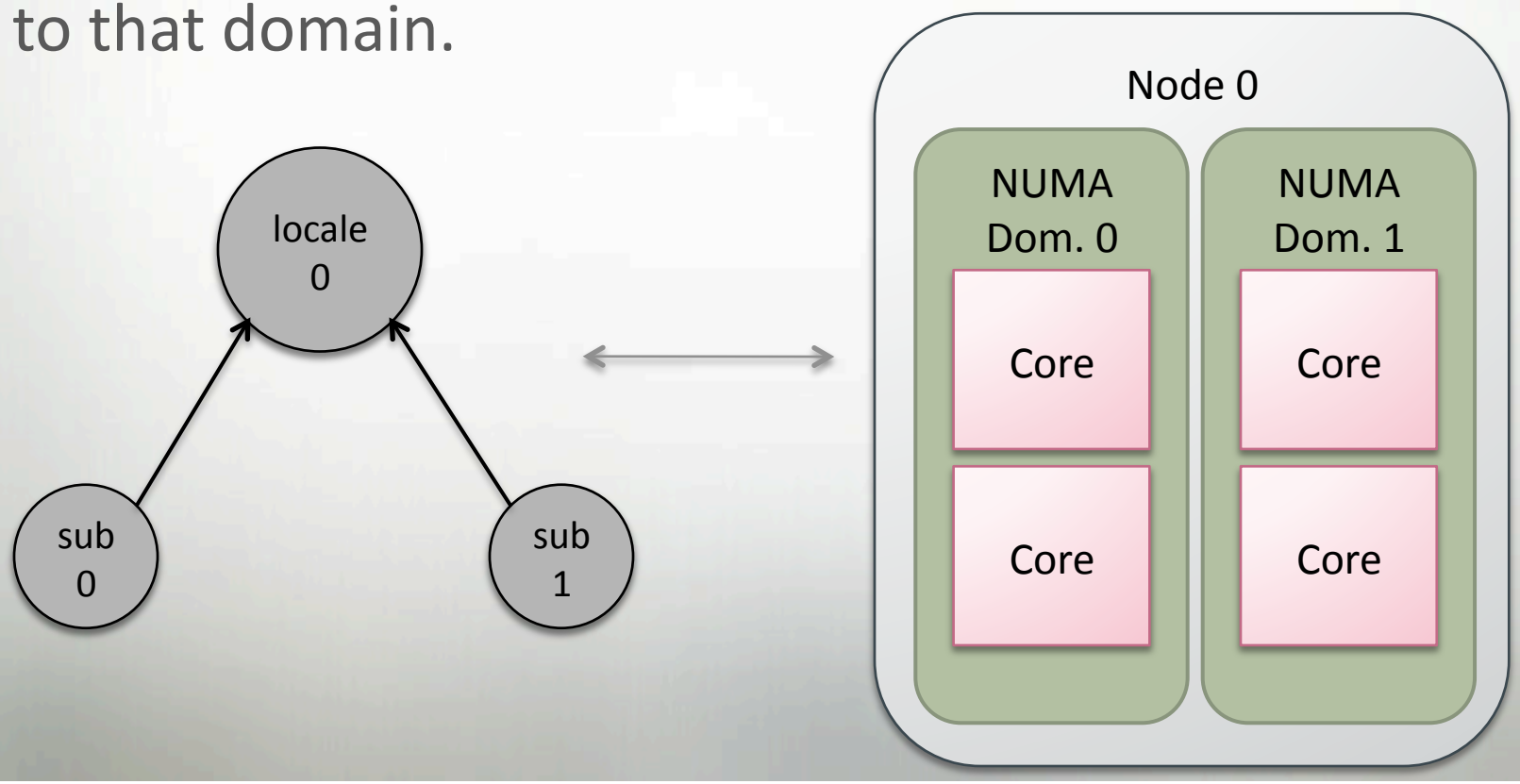
Case Study: STREAM Benchmark

- A = B + alpha * C, vectors generated randomly
- Stresses memory bandwidth (memory-bound computation)
- NUMA issues in current Chapel implementation:
  current Chapel implementation suffers due to lack of care w.r.t. NUMA placement

Hierarchical NUMA Model

Solution:

- Within a locale, define a "sublocale" for each NUMA domain.
- When initializing a task on a sublocale, assign affinity to that domain.



Hierarchical NUMA Model

- "*qthreads*" task layer defines "*shepherds*," which are thread-mobility domains.
- *qthreads* will not move threads outside of their assigned shepherd, if possible
- thus, assign a shepherd to each socket

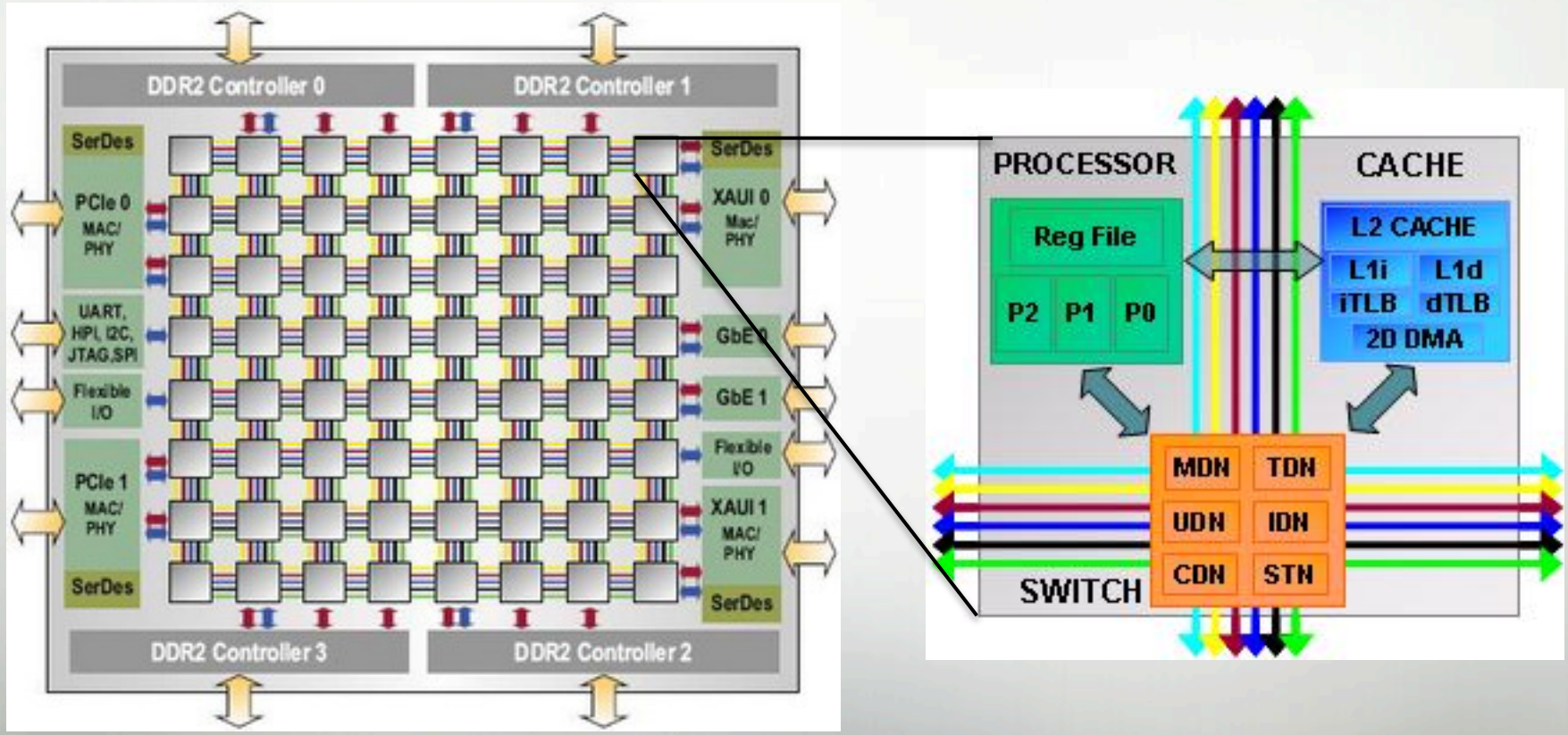
```
class NumaSocket : locale {
  var my_subloc_id : int;

  proc NumaSocket(cpu: int) {
    chpl_id = __primitive("chpl_localeID");
    numCores = numa_num_cores;
  }
  proc initTask() : void {
    qthread_migrate_to(my_subloc_id);
  }
...
```

Cray XE Benchmark

- STREAM Triad on hierarchical locales
- 16-core XE nodes, 4 shepherds, one task per shepherd
- 9.2 GB/s, Chapel baseline
- 10.9 GB/s, Chapel with qthreads tasking
- Pinning tasks to shepherds ensures task/data locality

Hierarchical Tilera Model

- Similar issue: we want an interface to associate work/memory with a particular tile.



Hierarchical Tilera Model

```
class TileraPart : locale {
  var my_cpus : cpu_set_t;

  // Associate this Tilera partition with a CPU set.
  proc TileraPart(part_desc:string) {
    tmc_cpus_from_string(my_cpus, part_desc);
  }
  // Set the affinity of each new task to my CPU set.
  proc initTask() : void {
    tmc_cpus_set_my_affinity(my_cpus); // Magic sauce.
  }
  ...
  ...
}
```


Tilera Benchmark

- **Baseline** SMP version lets OS assign tasks to cores
- **Sublocale** version maps data and tasks to specific cores

Tasks per Core	SMP	Sublocale
1	212 MBps	369 MBps
2	234 MBps	203 MBps

- Explicit mapping runs about 60% faster than the OS-default caching and task scheduling

Hierarchical Locales

- How to leverage hierarchical locales in your code?
- Simple: "**use**" the proper architecture module in your code. *i.e* **use** tilera, *or* **use** xe_numa
- Use "**on**" statements or pre-made distributions to target sublocales.

examples:

```
on loc.getChild(tid)
dmapped Block(boundingBox = ProblemSpace, targetLocales = sublocs);
```

Conclusion

- Hierarchical locales enable:
  - Detailed architectural descriptions
  - Locale-aware memory allocation and tasking
  - Opportunities for increased performance

Future Work

- To be merged with trunk:
  - Encapsulation of system architecture in *RootLocale*
  - Locale-aware memory allocation and tasking
  - Arch. modules for Tilera and Cray systems
- Apply to more interesting computations than STREAM
- Apply to other node architectures (CPU/GPU, Intel MIC, etc)

CRAY

THE SUPERCOMPUTER COMPANY