Surveying the PDC Landscape

Joel Adams
Calvin College

Overview

Let's explore two related-but-different areas:

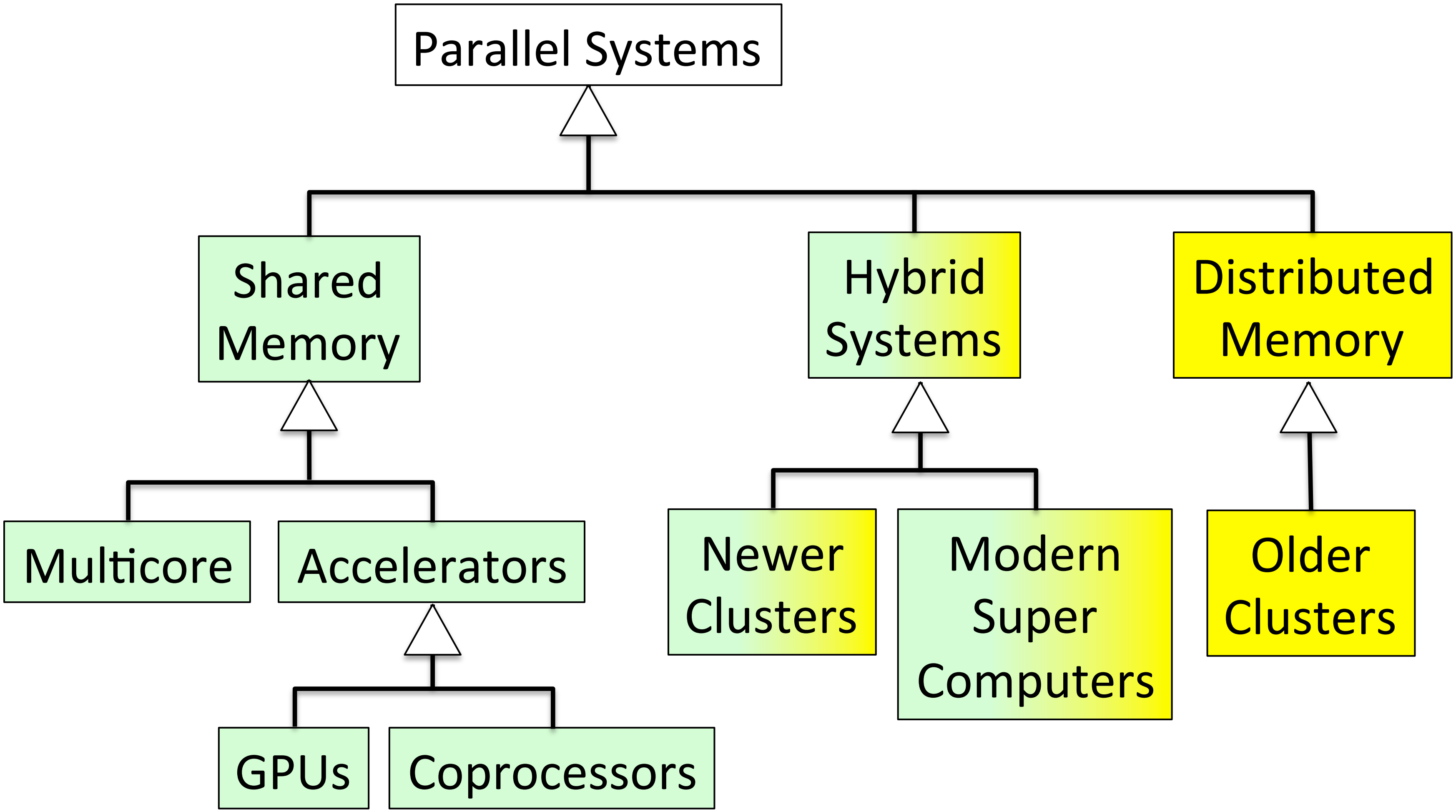
- Shifts in the hardware landscape
- Corresponding changes on the software side

But first…

A Word From Our Sponsors

- A CSinParallel "module" is 1-3 days worth of teaching materials on PDC topics, usually including hands-on activities/tutorials.
- The CSinParallel project includes $$ for module authors – PDC educators willing to share / convert their teaching materials into CSinParallel modules.
- If you are interested in *authoring*, talk to *me*!

Hardware Landscape



History / Timeline

Decade	PDC Hardware Platforms	Memory
1980s	Vector supercomputers	Shared
	Multiprocessors (networked)	Distributed
1990s	Cluster supercomputers	Distributed
	Internet	Distributed
	Symmetric multiprocessors	Shared
2000s	GPUs	Shared
	Multicore processors	Shared
2010s	Hybrid supercomputers/clusters	Both
	Coprocessors (w. vector units)	Shared

History / Timeline

Decade	PDC Hardware Platforms	Software
1980s	Vector supercomputers	Proprietary
	Multiprocessors (networked)	Proprietary
1990s	Cluster supercomputers	MPI
	Internet	Sockets,BOINC
	Symmetric multiprocessors	Various
2000s	GPUs	CUDA,OpenCL
	Multicore processors	Various
2010s	Hybrid systems	Combinations
	Coprocessors (w. vector units)	Various

Today's Software Landscape

- The software generally varies with the hardware platform it is intended to run on:
  - Distributed memory systems
  - Shared memory systems
    - Vanilla shared-memory systems
    - Shared-memory systems with Accelerators
      - Manycore GPUs and/or coprocessors
  - Hybrid systems
- No standard "one size fits all" solution (yet)
- Let's explore these one at a time...

Distributed Memory Systems

Two broad categories; both use standalone *compute nodes*, each with their own memory:

- Local-area distributed-memory systems
  – Nodes are connected via a *local area network* (the faster the better)
- Wide-area distributed-memory systems
  – Nodes are connected via a *wide area network* (such as the Internet – comparatively slow)

Distributed Memory System Software

Local-area dist-mem. systems use *multiprocessing*:

- Remote *processes* are launched on compute nodes
- The message passing interface (MPI) is the industry standard platform for such systems
  - Implementations for C, C++, Fortran, Python, …
  - Generality: Works well on *shared-memory systems* too
- MapReduce is a Google platform for *reliably* solving some kinds of distributed problems
  - Hadoop is an open-source version of MapReduce
  - WebMapReduce is a browser-based Hadoop front end developed by Dick Brown + St. Olaf students

MACALESTER COLLEGE

CALVIN
MINDS IN THE MAKING

ST·OLAF COLLEGE

Distributed Mem. System Software (2)

For *wide-area* distributed-memory systems:

- Remote processes communicate via *sockets*:
  - Client-server systems are most common
  - Peer to peer systems are a decentralized alternative…
- The Berkeley Open Infrastructure for Network Computing (BOINC) is a widely used platform for coordinating distributed computing tasks:
  - SETI@Home, Folding@Home, LHC@Home, …

The relative slowness of wide-area communication limits this approach to embarrassingly parallel problems.

Shared Memory Systems

Three broad categories:

- Vanilla shared-memory systems
  - Multicore / Multisocket CPU-based systems
  - Cores share a common memory
- Accelerated shared-memory systems
  - Vanilla systems plus many-core accelerator(s) (GPGPU, Coprocessor)
- "Hybrid" systems: CPU+GPU on same chip

Shared-memory systems use *multithreading*

Vanilla Shared Mem. Software

Vanilla shared-memory systems are ubiquitous:

- Open MultiProcessing (OpenMP) is an industry standard for multithreading
  - Non-proprietary open standard
  - Multilanguage support (C, C++, Fortran)
  - Pragma-based programming; relatively easy
- Lots of language-based multithreading options:
  - Java, C (pthreads), C++11 (Boost), C# (.NET), …
- Vendor-specific (proprietary) libraries/languages
  - Intel's Thread Building Blocks (TBB), Google's Go, …

iPhone 5: dual-core A6 chip

iPad 3: quad-core A5X chip

Vanilla Shared Mem. Software (2)

Vanilla shared-memory systems can also be programmed via *message-passing*:

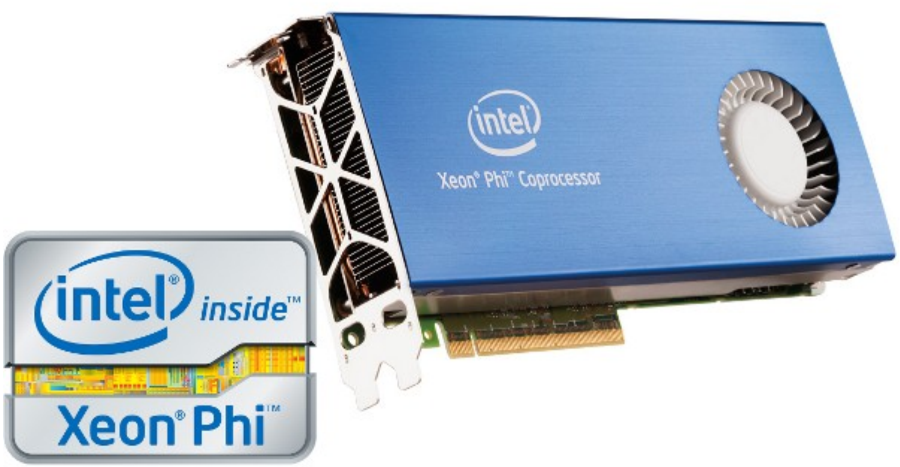
- MPI also works well on these systems
- Some languages utilize *message-passing tasks* to avoid multithreading's *race conditions*:
  - Erlang, Scala, …
  - Programs written in these languages port easily to distributed-memory systems

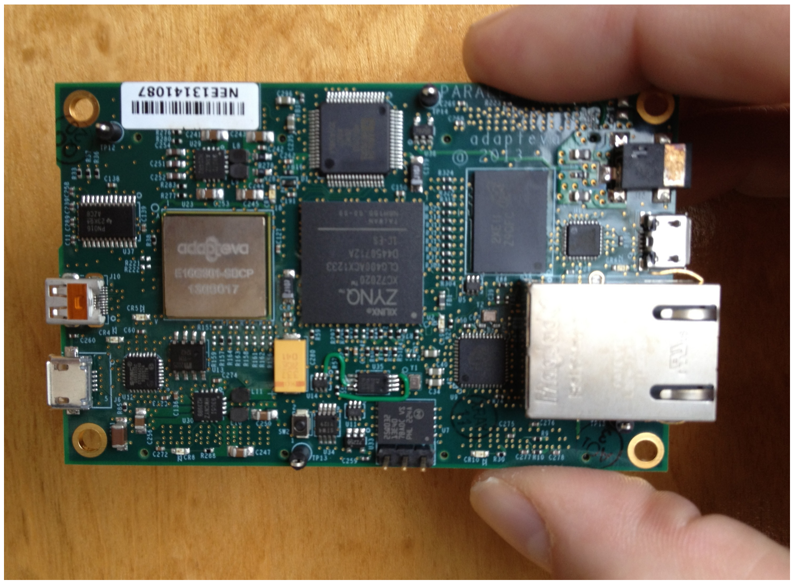
Every CS undergraduate student should learn how to program vanilla shared-memory parallel systems

Accelerated Shared Mem. Systems

Software for shared-memory systems with accelerators varies with the accelerator:

- General Purpose Graphics Processing Unit (GPU) systems
  - Nvidia
  - AMD's ATI/Radeon
- Coprocessor systems
  - Intel's *Xeon Phi* (61 cores, 4 hw threads/core), available to us on Intel's Manycore Testing Lab (MTL)
  - Parallela's *Epiphany* (16 cores)





MACALESTER COLLEGE

CALVIN
MINDS IN THE MAKING

ST · OLAF
COLLEGE

Accelerated Shared Mem. Software

Software for shared-memory systems with *GPUs*:

- Compute Unified Device Architecture (CUDA):
  - – Proprietary; works only on Nvidia GPU cores
  - + Well established; extensive examples/documentation available
- Open Compute Language (OpenCL):
  - + Platform independent open standard
  - + Can use every core in a system (Nvidia or not)
  - – Significantly more complicated than CUDA
  - – Fewer examples/tutorials/documentation available
- OpenACC (Open Acceleration?):
  - + Pragmas (a la OpenMP) to simplify GPU computing
  - – Promising, but still in development
- Intel's Array Building Blocks (ArBB):
  - + C++ library for vectorized parallel computing
  - – Proprietary; C++ only


nVIDIA
TESLA

Accelerated Shared Mem. Software (2)

Shared-memory systems with *coprocessors* (cluster on a chip):

- MPI
- OpenMP
- OpenCL
- Intel's ArBB

Coprocessors are fairly new, so other software platforms for them will likely appear…

Hybrid Systems

Complications:

1. Unicore CPUs are just about extinct…
   – All recent clusters have multicore CPUs
2. Accelerators can be added to cluster nodes

This creates lots of hybrid-system options

– Distributed + shared memory

– Distributed + shared memory + GPUs

– Distributed + shared memory + coprocessors

Hybrid System Software

- Distributed + shared memory
  - MPI (1 MPI process/core)
  - MPI + OpenMP (1 MPI process/node)
  - MapReduce (1 or more MR process/node)
- Distributed + shared memory + GPUs
  - MPI + CUDA
  - MPI + OpenMP + CUDA
  - MPI + OpenCL
- Distributed + shared memory + coprocessors
  - MPI
  - MPI + OpenMP

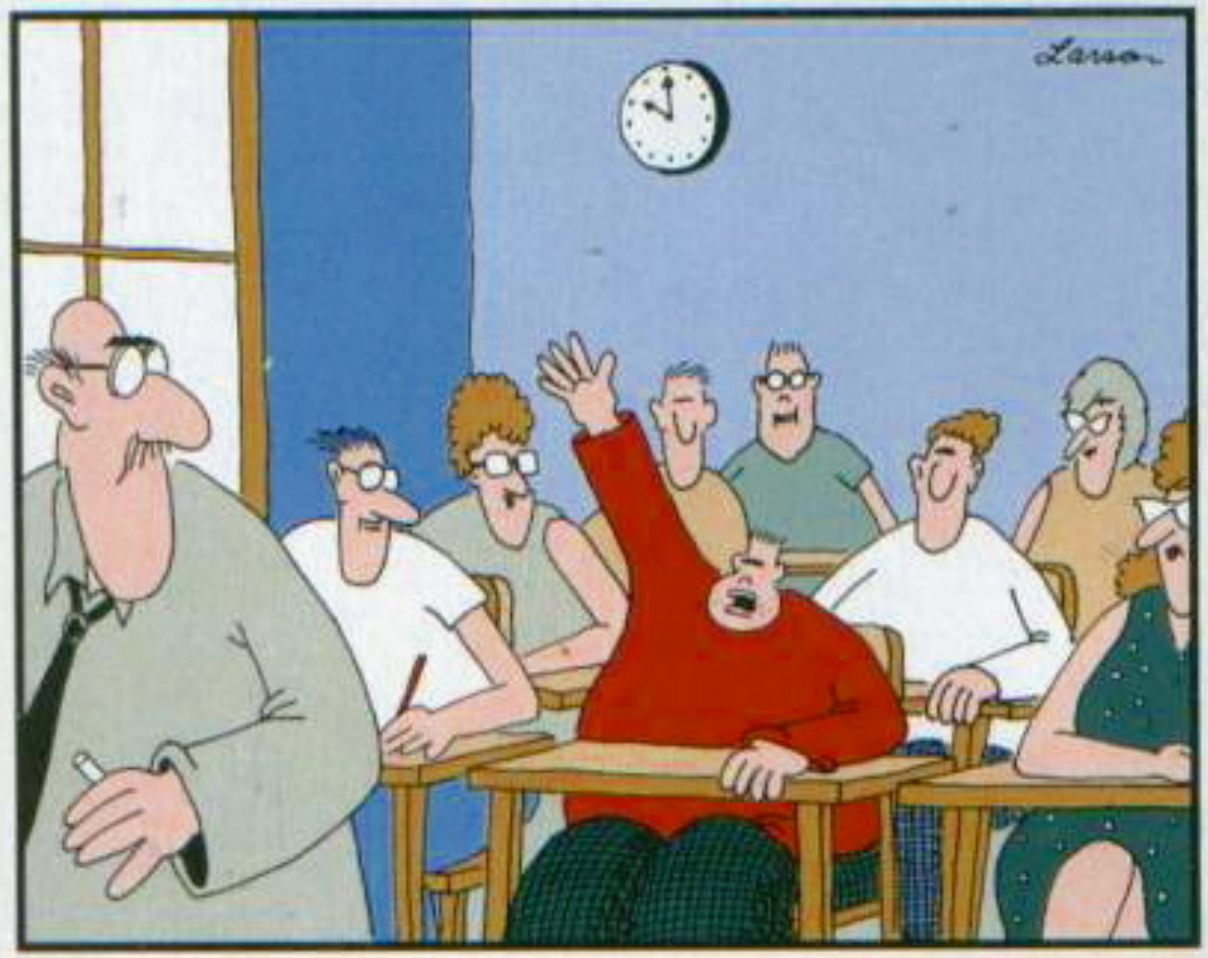
Problems

- MPI, OpenMP, etc have carried us this far, but the experts say they are *insufficient* to let us reach *exascale* computing
- MPI is relatively low-level
- Newer high level languages are being developed to make it easier to develop scalable programs (at least for distributed+shared mem. hybrids):
  - Scala: immutable OO Actors, message passing, JVM
  - Chapel: APGAS language from Cray
  - …

APGAS

… asynchronous partitioned global address space

- All tasks share a global address space / memory
- All tasks can access the entire space, but the address space is logically partitioned, so a task may have *affinity* for a particular partition:
  - Thread-local memory on a shared mem. system
  - Process memory on a distributed mem. system
  - …
- Merges strengths of shared+distributed systems
  - Chapel, Unified Parallel C (UPC), X10, Fortress, …



"Mr. Osborne, may I be excused? My brain is full."

Information Overload

- If you are saying to yourself:
  - *"This is overwhelming!"*
  - *"There's no way I can learn all of this stuff."*
  - *"PDC is so unstable; is there any content that is worth my time to learn / not ephemeral?"*

  Don't feel bad; you're not alone!
- One of our goals is to establish a community to provide mutual support for PDC educators!

Parallel Patterns

- … are industry-standard practices and techniques that have proven useful in many different parallel contexts.
- … are built into popular platforms like MPI and OpenMP.
- … seem likely to have long-term usefulness, regardless of future PDC developments.
- … provide a way to organize PDC concepts.

Parallel Patterns (2)

Example 1: Most parallel programs use one of just three *parallel algorithm strategy patterns*:

- Data decomposition: processes/threads divide up the data and process it in parallel.
- Task decomposition: processes/threads divide the algorithm into functional tasks that they perform in parallel (to the extent possible).
- Pipeline: processes/threads divide the algorithm into linear stages, through which they "pump" the data.

Parallel Patterns (3)

- Data decomposition:

Thread 0

Thread 1

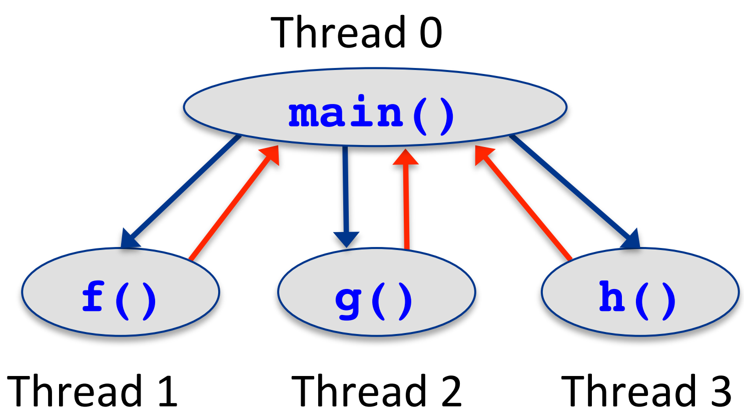
Thread 2

Thread 3

Parallel Patterns (4)

- Task decomposition:
  - The independent functions in a sequential computation can be "parallelized":

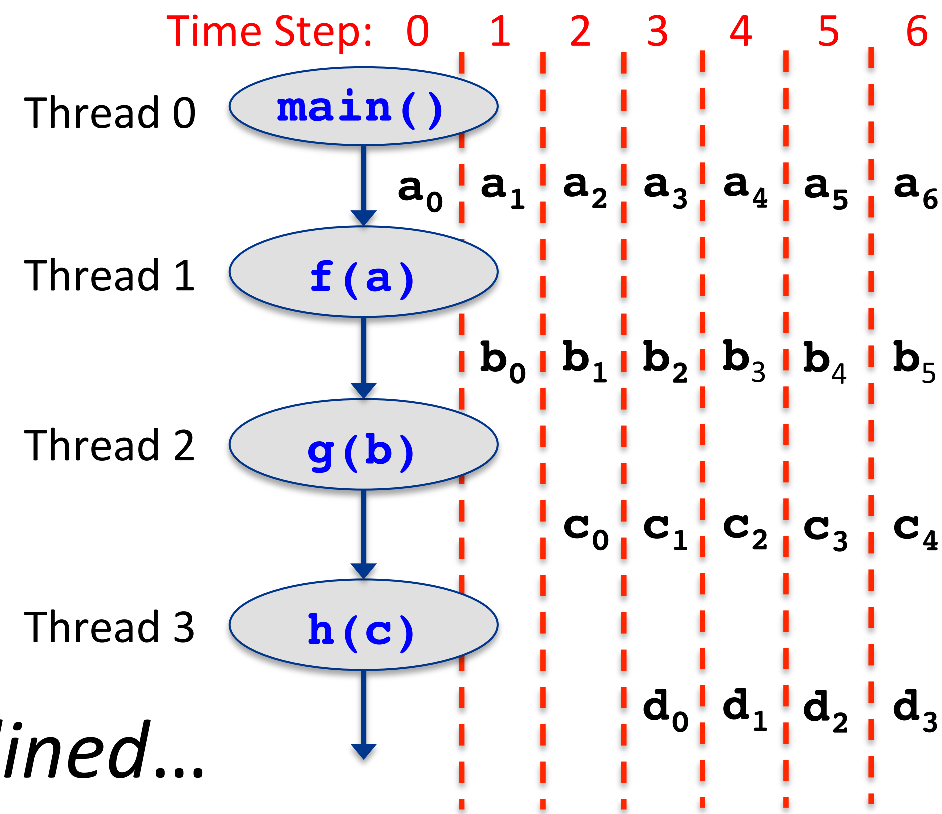
```
int main() {
    x = f();
    y = g();
    z = h();
    w = x + y + z;
}
```



Parallel Patterns (5)

- Pipeline: When functions are not independent:

```
int main() {
    ...
    while (fin) {
        fin >> a;
        b = f(a);
        c = g(b);
        d = h(c);
        fout << d;
    }
    ...
}
```

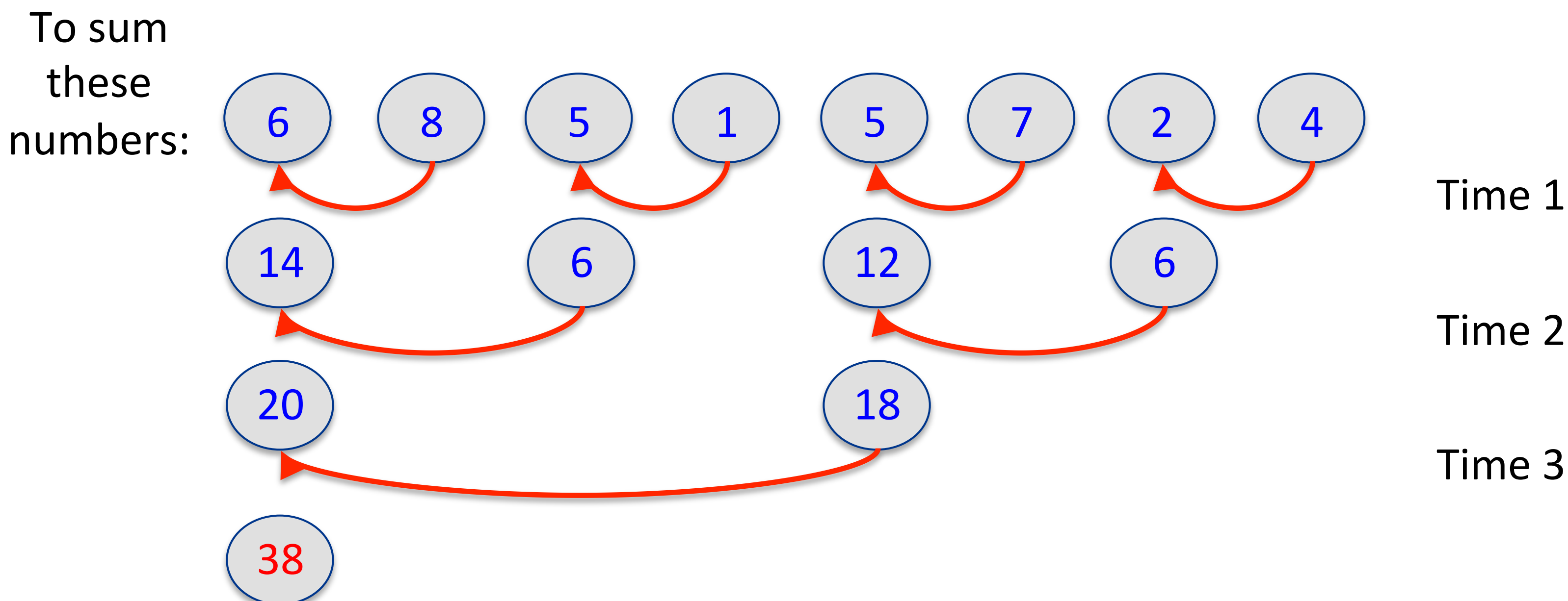


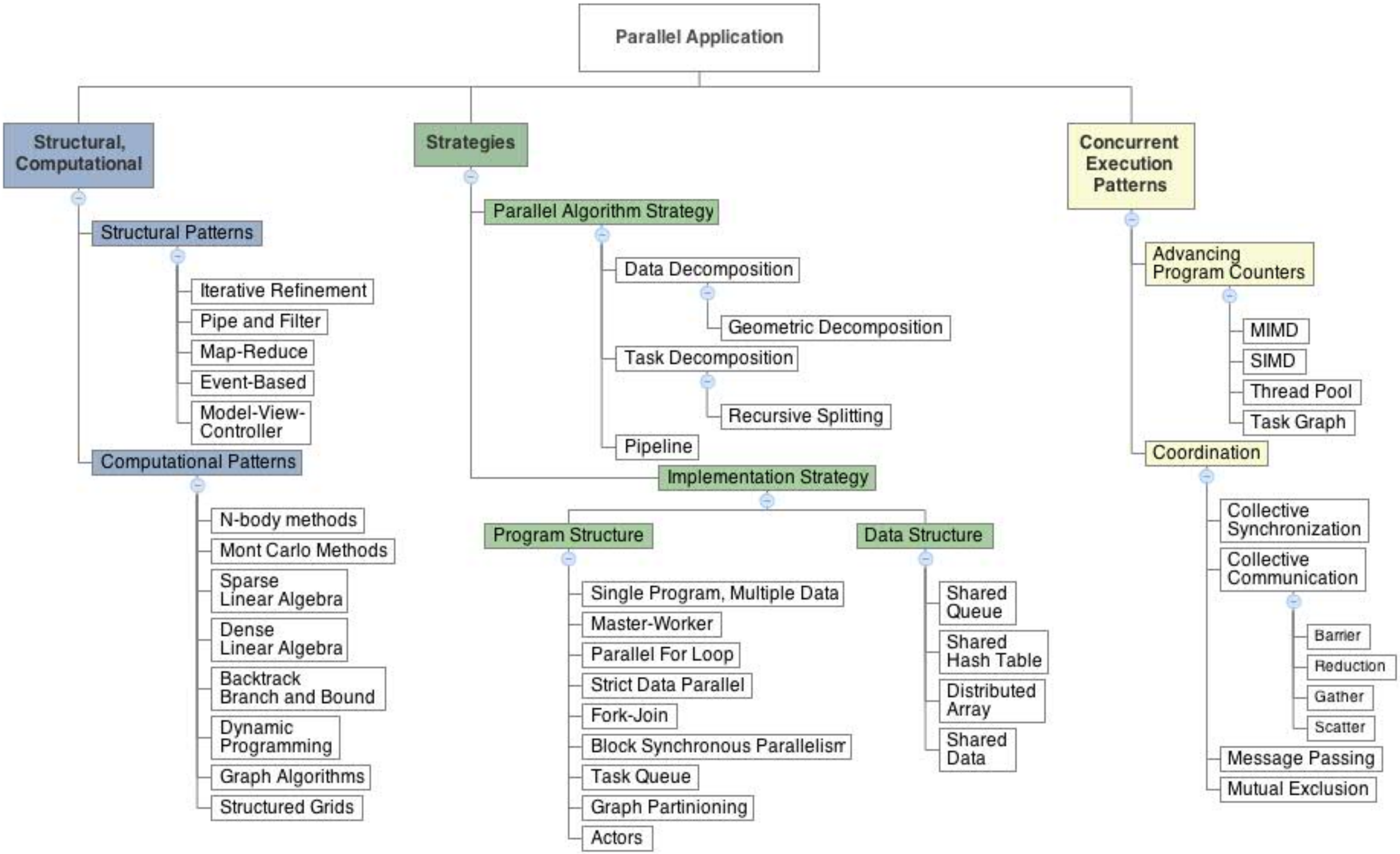
they can still be *pipelined*…

Parallel Patterns (6)

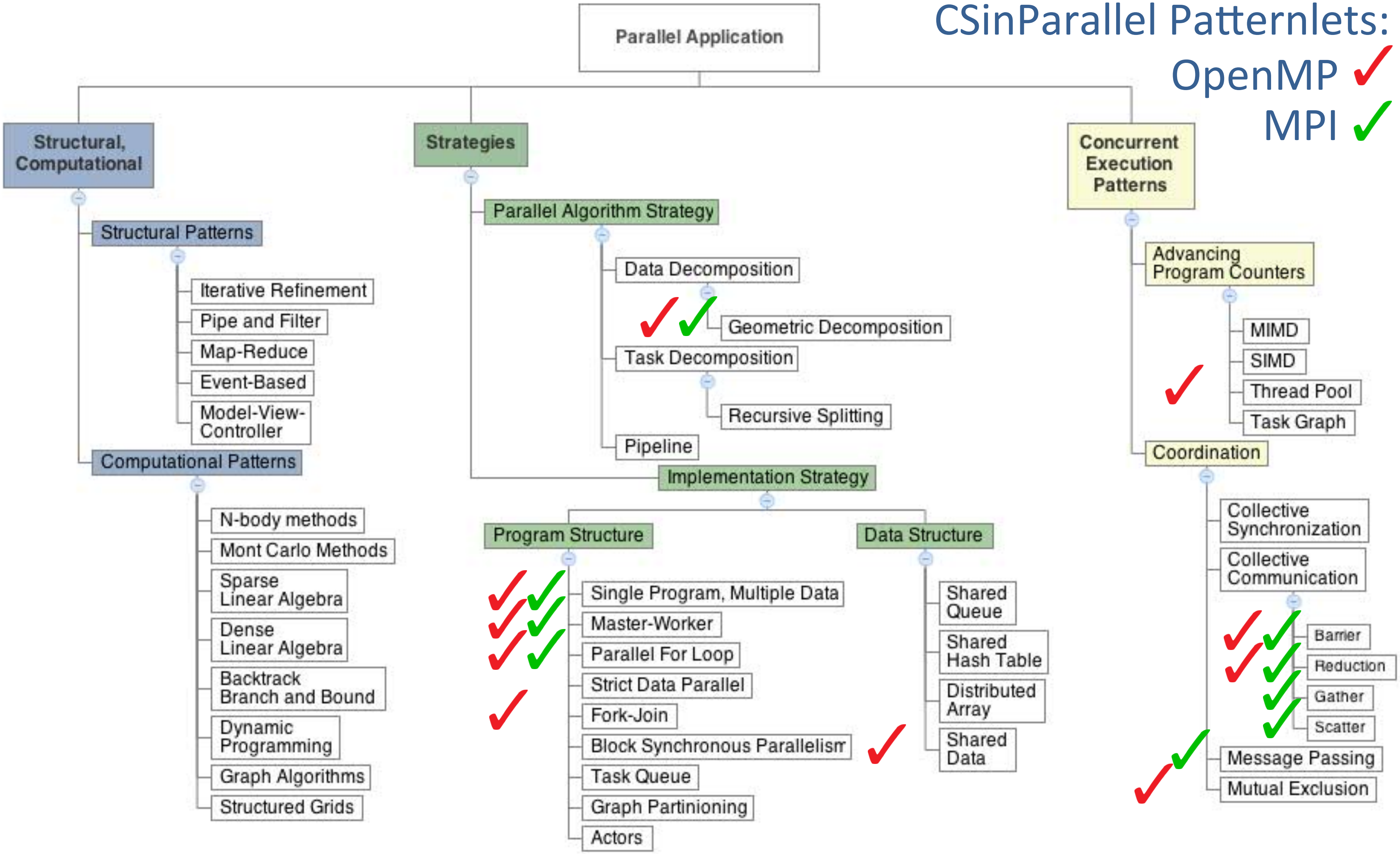
Example 2: Parallel programs often need to combine the local results of N parallel tasks.

- When N is in the millions, *O(N)* time is too slow
- The reduction pattern does it in *O(lg(N))* time:




Parallel Application
Structural, Computational
Structural Patterns
Iterative Refinement
Pipe and Filter
Map-Reduce
Event-Based
Model-View-Controller
Computational Patterns
N-body methods
Mont Carlo Methods
Sparse Linear Algebra
Dense Linear Algebra
Backtrack Branch and Bound
Dynamic Programming
Graph Algorithms
Structured Grids
Strategies
Parallel Algorithm Strategy
Data Decomposition
Geometric Decomposition
Task Decomposition
Recursive Splitting
Pipeline
Implementation Strategy
Program Structure
Single Program, Multiple Data
Master-Worker
Parallel For Loop
Strict Data Parallel
Fork-Join
Block Synchronous Parallelism
Task Queue
Graph Partinioning
Actors
Data Structure
Shared Queue
Shared Hash Table
Distributed Array
Shared Data
Concurrent Execution Patterns
Advancing Program Counters
MIMD
SIMD
Thread Pool
Task Graph
Coordination
Collective Synchronization
Collective Communication
Barrier
Reduction
Gather
Scatter
Message Passing
Mutual Exclusion

CSinParallel Patternlets:
OpenMP ✓
MPI ✓



Patternlets…

… are minimalist, scalable, executable programs, each illustrating a particular pattern's behavior:

- *Minimalist* so that students can grasp the concept without non-essential details getting in the way
- *Scalable* so that students see different behaviors as the number of threads changes
- *Executable* to let students see the pattern in action:
  - Instructors can use it in a live-coding demo
  - Students can use it in a hands-on exercise

We encourage you to explore the (still growing) CSinParallel Patternlets module.

Conclusions

- You have to start somewhere
  - Getting started is more important than where
- PDC hardware resources are abundant
- Choose a software platform(s) based on what will work best at your institution/department:
  - C/C++: MPI+OpenMP
  - Language agnostic: Chapel
  - Java: Scala
  - …
- CSinParallel is here to help!

Thank You!