Parasol
Smarter computing.
Texas A&M University

Write clean (parallel) code!

Bjarne Stroustrup

Texas A&M University

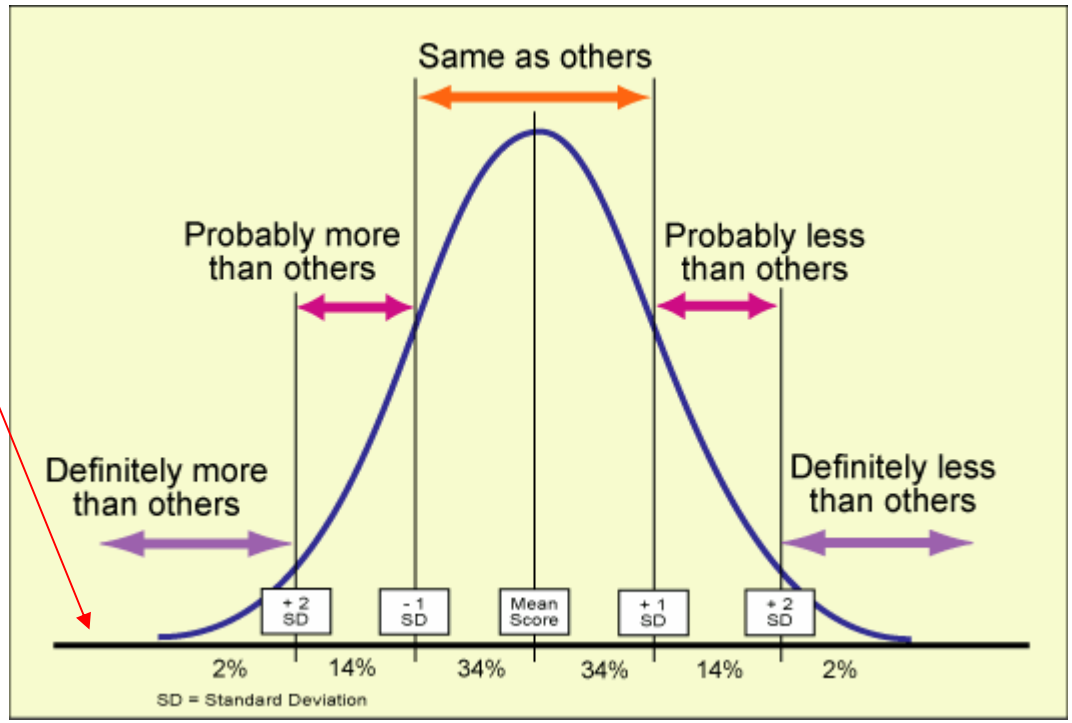
http://www.research.att.com/~bs

Assumptions

- You know more about architecture and optimizing parallel code than I do
- A keynote articulates ideals
  - Supported by reason
  - Questions are usually more important than answers
  - (Must not show code)
  - (Must have fancy graphics)
- We want to see parallel and distributed programs in mainstream use
  - Very soon (1 year, 2 years, 5 years)
  - We have had about 50 years of research, let's use it

Take pity on us

- You are here



- We'll have 10,000,000 programmers
- Just 50% of all programmers are above average

Clean code

Parasol
Smarter computing.
Texas A&M University

- Is easy to read and write
  - Provided you know the application domain
  - Maybe the application does require a Ph.D.
    - But it has better not be a Ph.D. in Computer Science
- Is easy to debug
  - For some definition of “easy”
- Is easy to maintain
- Is easy to port
- Runs efficiently
- Does not contain performance viruses

A performance virus

- Is a piece of code that runs well, but after a small change or a port runs abysmally, e.g.
  - An **O(n*n)** algorithm for a small **n** in a test run
  - A program with a hardwired constant reflecting a critical cache size
  - A program written assuming a shared memory naively ported to a cluster (or vise versa)
  - A program assuming exactly 6 processors
  - …
- A non-portable program is a performance virus as long as hardware performance keep improving

New languages

Parasol
Smarter computing.
Texas A&M University

- Maybe clean parallel code requires a new language
  - DARPA thinks so (?): Fortress, X10, ???
  - Academics always think so
  - But maybe not, a new language
    - typically dies without helping the user community solve problems
      - When it's designer graduates, gets tenure, or is promoted
      - It's company gets a new CEO
    - consumes resources which could have spent elsewhere
      - Design, implementation, learning to use
    - won't be useful until about 5 years (or more) after the project starts
    - aimed at parallelism is unlikely to be competitive with existing languages for non-parallel code
      - For years
      - And non-parallel code will be most code for a (long) while

What is simple enough?

Parasol
Smarter computing.
Texas A&M University

- Perfect automatic parallelization of arbitrary applications
  - Sure, but impossible
- "I want this to be parallel, but spare me the details"
  - What, not how
  - Leave "ordinary code" untouched
- Provide "ordinary users" with parallel components
  - Into which they can plug their own code
  - That they can drop into their existing applications
  - Try to verify that the user code doesn't mess with shared data
- "Threads and locks" programming is evil
  - Breeds complexity
  - Too complex for "ordinary programmers"
  - Required in real-world code
  - We need all the (language and tool) help we can get

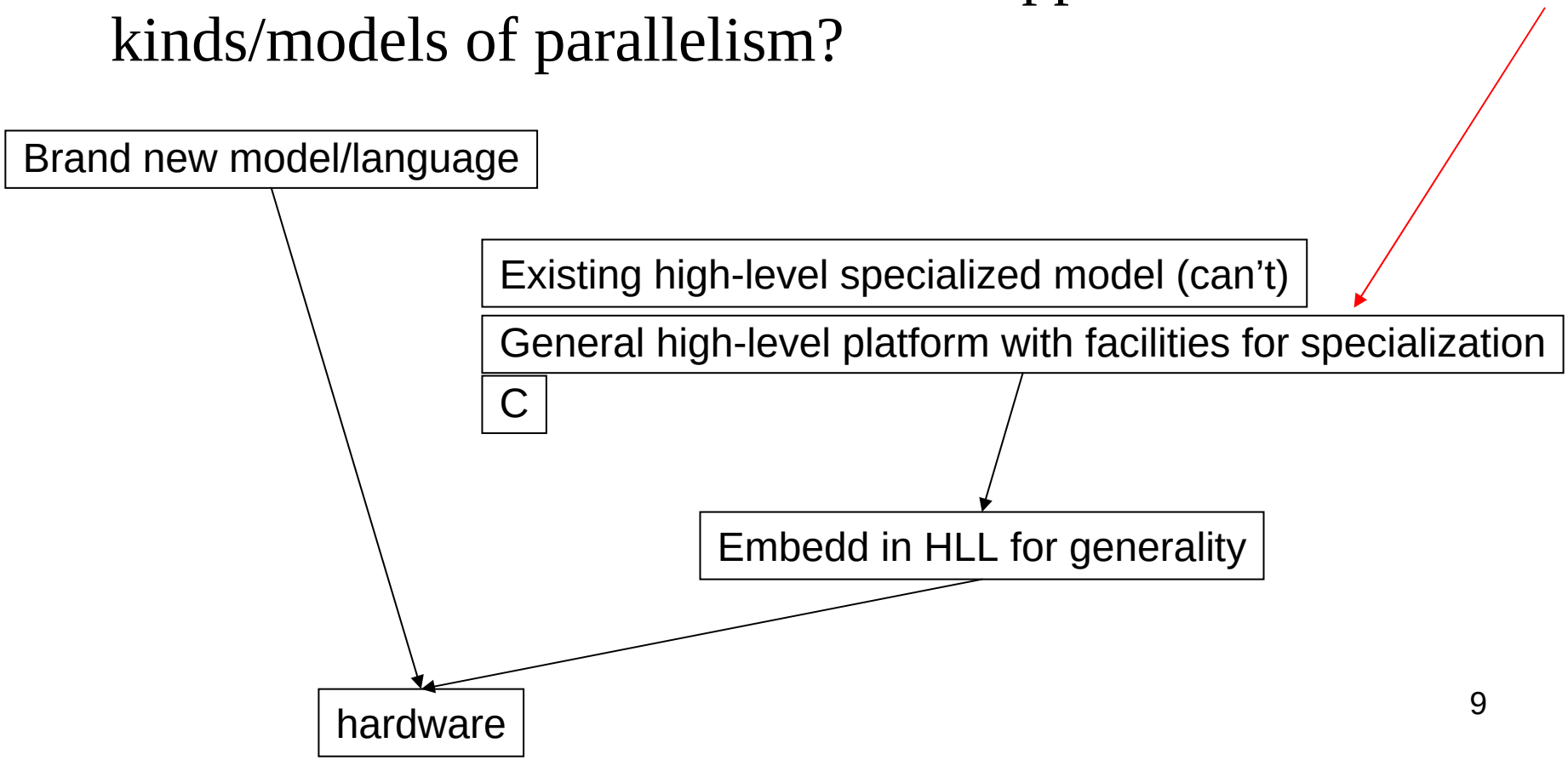
Is there enough parallelism?

Parasol
Smarter computing.
Texas A&M University

- Traditional data parallel code has been scientific and numeric
- Most code is neither
- We must expand the application areas for parallel techniques
  - Data mining/analysis
    - Finance
    - Advertising
  - Graphics
  - Everyday tasks in parallel
    - Compilation
    - Typesetting
    - Regression testing
- Each application area for parallel techniques cannot have its own language, tools, and techniques
  - Currently, it seems that every field is (re?) inventing the wheel(s)
- We must integrate data parallel techniques in languages and general-purpose tool chains

Is one kind of parallelism sufficient?

- Of course not
- From what level should we build support for new kinds/models of parallelism?



What would *I* like to see?

Parasol
Smarter computing.
Texas A&M University

- Nah, sure, if you must, someone has to do it
  - 10% improved cache performance?
  - 1% extra performance on each of 256 processors
  - Using 1,000,000 processors for heat transfer computations
  - Real-time rendering of blood, gore, and porn
- Yes!
  - “two simultaneous troffs”
  - Compiling 30 source files simultaneously on “an ordinary PC”
  - Running 20,000 regression tests 1,000-way parallel
  - Indexing the web in a couple of hours
  - 10* speed up in Photoshop real-time rendering
- That is:
  - By all means research the most advanced cases,
  - But don’t forget what’s soon to be “the low end”

Keep Simple Things Simple

Parasol
Smarter computing.
Texas A&M University

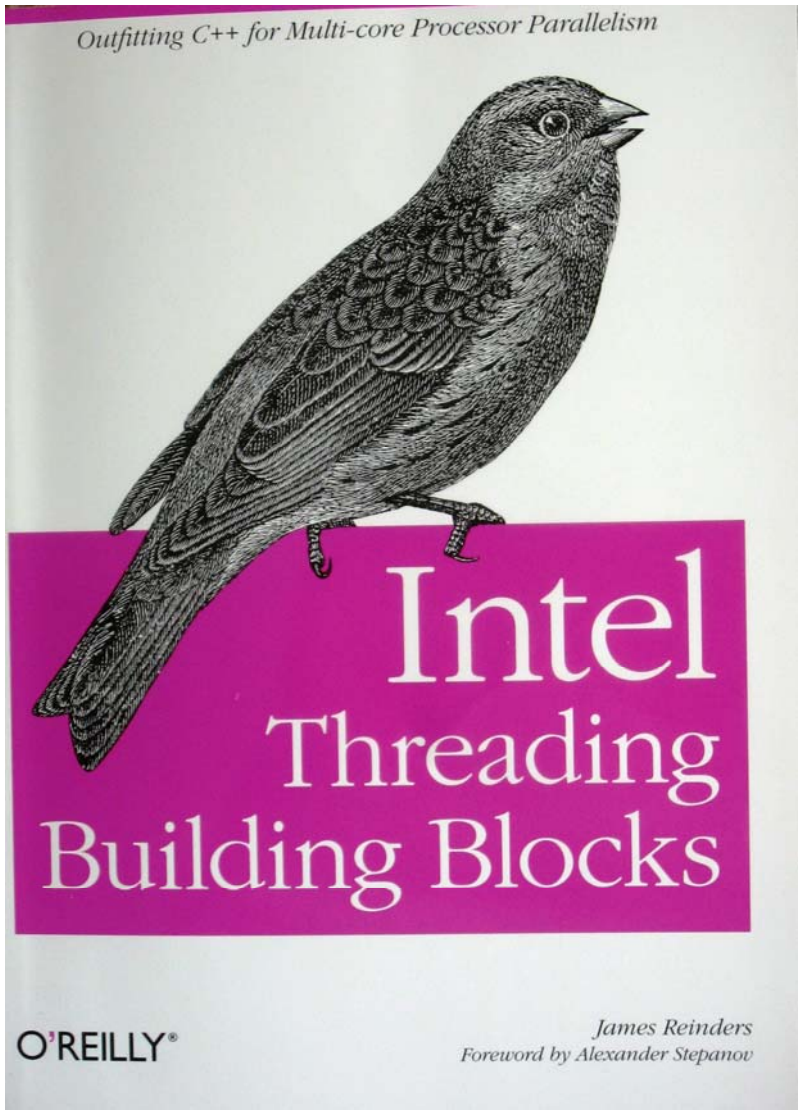
- Not KISS
  - It's not stupid – it's really hard
  - Only simple things can be simple
    - That's no excuse for making everything complicated
      - Defaults!
      - Tools
      - Design for usability
  - Apply recursively
    - Everything should be no more complicated than necessary

Two examples

- STAPL
  - TAMU Standard Template Adaptive Parallel Library
  - Main designer: Lawrence Rauchwerger
  - http://parasol.tamu.edu/groups/rwergergroup/research/stapl/
- TBB
  - Intel Threading Building Blocks
  - Main designer: Arch Robinson
  - http://osstbb.intel.com/

- Both C++ and STL inspired
  - Neither perfect (so far)

TBB



- Main designer
  - Arch Robison
- Builds on the work of many friends
  - Alex Stepanov
  - Lawrence Rauchwerger
  - …
- Builds on principles I strongly support
  - Library building
  - Composition of code
  - Lightweight concurrency

TBB quotes

- Stepanov
  - Building libraries is an important task
  - Non-intrusive, coexist, orthogonal, not hide useful information, efficient
- Robison
  - Parallel programming is no longer optional
  - There is no *one true way* to do parallel programming
  - Separate logical tasks from physical threads
  - Strictly a library
  - Recursive parallelism (based on ranges)
  - Performance matters

TBB

Parasol
Smarter computing.
Texas A&M University

- STL inspired (among others)
  - Templates
  - Function objects
  - Iterators (serial inspired) and ranges (necessary for parallelism)
  - RAII
- Coexistence with system threads
- Task based, not directly thread based
  - (relatively) cheap startup and join
    - (18 to 200 times faster than threads)
- Parallel algorithms
- Parallel containers
- …

Parallel algorithms

Parasol
Smarter computing.
Texas A&M University

- Currenctly supplied:
  - **parallel_for()**
  - **parallel_reduce()**
  - **parallel_scan()**
  - **parallel_while()**
  - **parallel_sort()**
  - **pipeline()**
- Each with ways of controlling grain/split
  - Default/key: recursive divide-and-concur

Parallel algorithms

- Rely on function objects

```
class SumFoo {
   float* my_a;
public:
   float sum;

   void operator()(const blocked_range<size_t>& r) {
            float *a = my_a;
            for (size_t i=r.begin(); i!=r.end(); +=i)
                     sum += Foo(a[i]);
            }

   SumFoo(SumFoo& x, split) : my_a(x.my.a), sum(0) { }

   void join(const SumFoo& y) { sum += y.sum; }
};
```

Parallel containers

Parasol
Smarter computing.
Texas A&M University

- "Clones" of the most useful STL containers
  - **concurrent_vector<T>**
  - **concurrent_hash_map<T>**
  - **concurrent_queue<T>**

- (too few – see STAPL)
  - A *consistent* set of algorithms or a *consistent* set of containers do not increase complexity
    - Irregularity does

Simple enough?

- Not yet
  - That doesn't mean that I know how to do it better
- To get mainstream use
  - Ordinary concepts and algorithms has to be "as simple as the textbook"
  - Parallel mechanism has to be "simpler than the textbook"
    - Elaboration/optimization is then possible
    - Parameterization with defaults plus tools
- Overuse of (C/Fortran built-in) arrays in interfaces
  - I'd like an Array<T,N> type
    - With all the usual arithmetic operations

Simple enough?

- Some of it is C++'s fault (*my* fault)
- C++0x will provide
  - Tasks
  - Atomic types
  - Concepts
    - A type system combinations of types and integers
  - auto (etc.)
    - deduce type from initializer (and other notational improvements)
  - Uniform and more flexible initialization
  - Move semantics
    - Decrease the cost of copying
  - Lambdas
    - Maybe
  - Attribute syntax
    - Dangerous (MPI)

Simple enough?

- Some of it is the expert's fault (*your* fault)
- Experts
  - Focus on the hardest problems
  - Focus on details
  - Write academic papers (only)
  - The cult of completeness

Complete enough?

- Completeness will come with time
  - Will it destroy simplicity?
  - It must not!

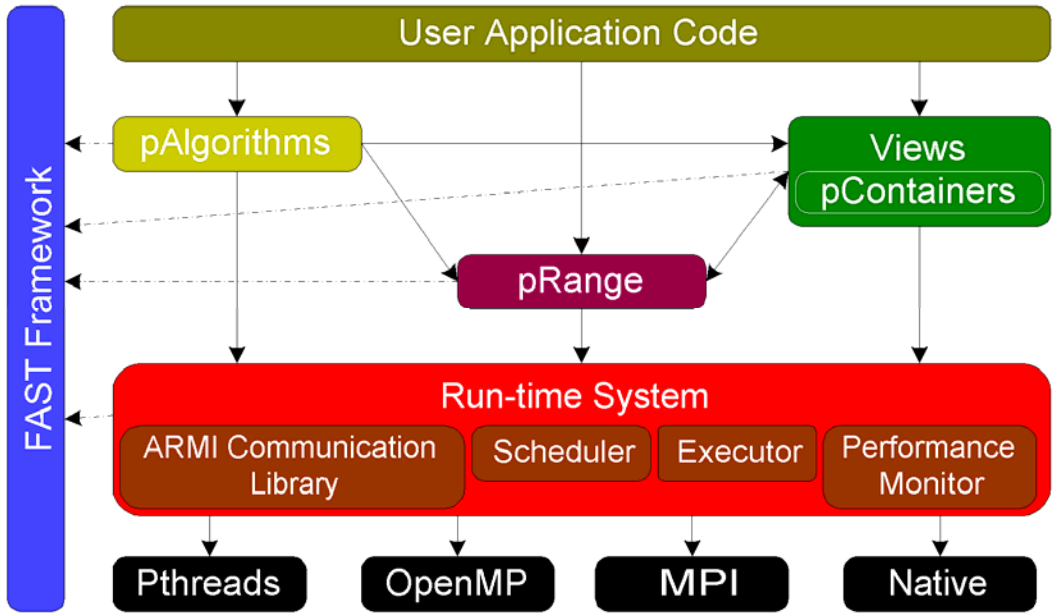
- Let's look at STAPL
  - A more mature/complete system
  - Distinction
    - Application developer (e.g. physicist)
      - Parallel/global-address-space/algorithmic/dependency world
    - Library developer (computer scientist)
      - Distributed/threaded world

Parasol
Smarter computing.
Texas A&M University

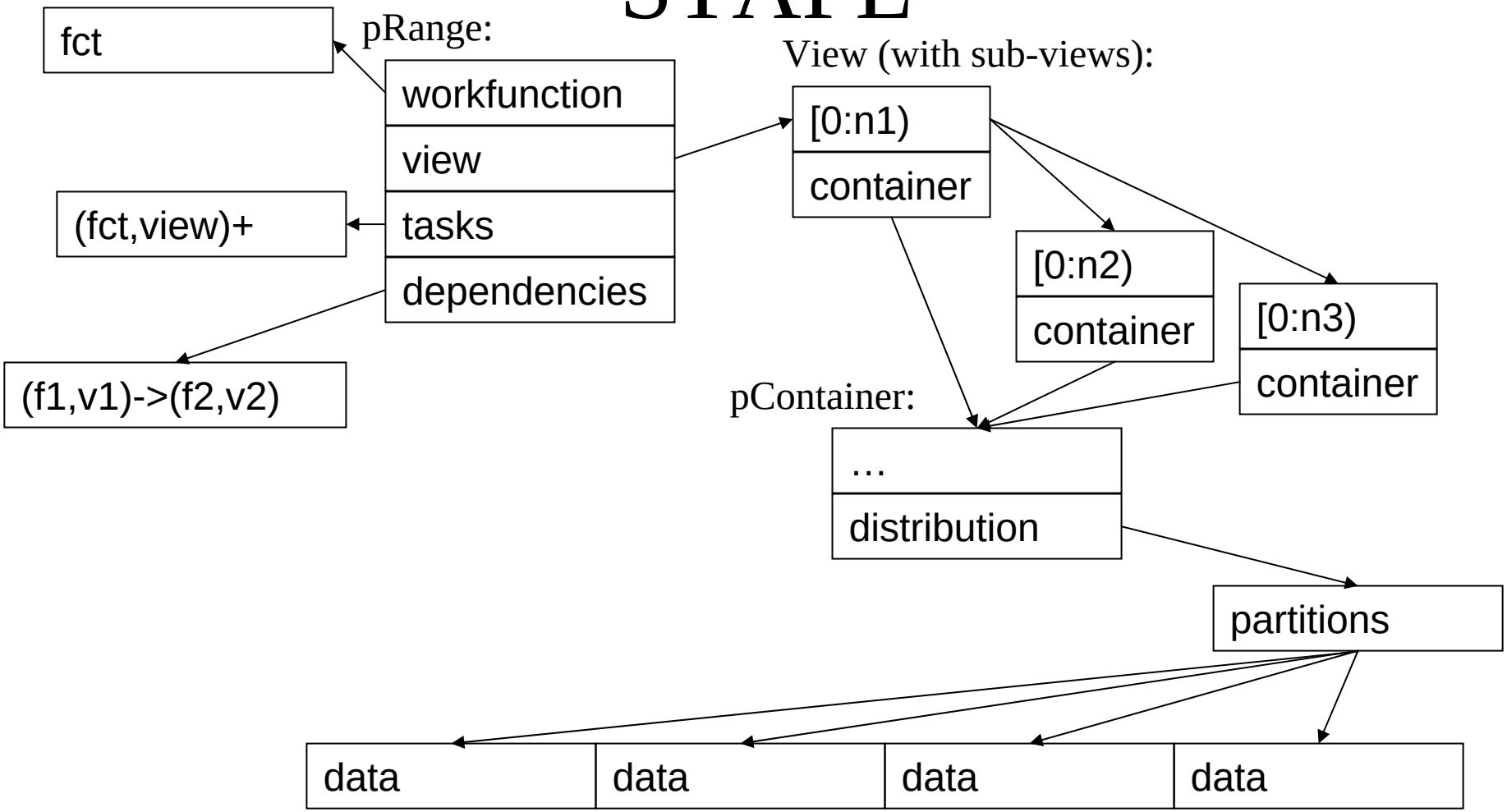
STAPL: Standard Template Adaptive Parallel Library

STAPL: A library of parallel, generic constructs based on the C++ Standard Template Library (STL)

- **Components for Program Development**
  - pAlgorithms, pContainers, Views, pRange
- **Portability and Optimization**
  - STAPL RTS and Adaptive Remote Method Invocation (ARMI) Communication Library
  - Framework for Algorithm Selection and Tuning (FAST)

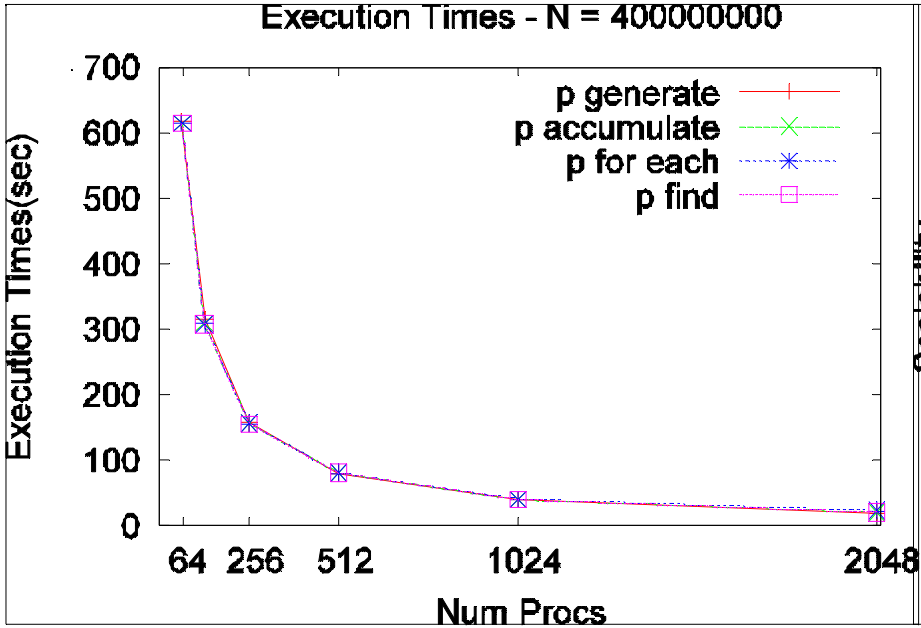


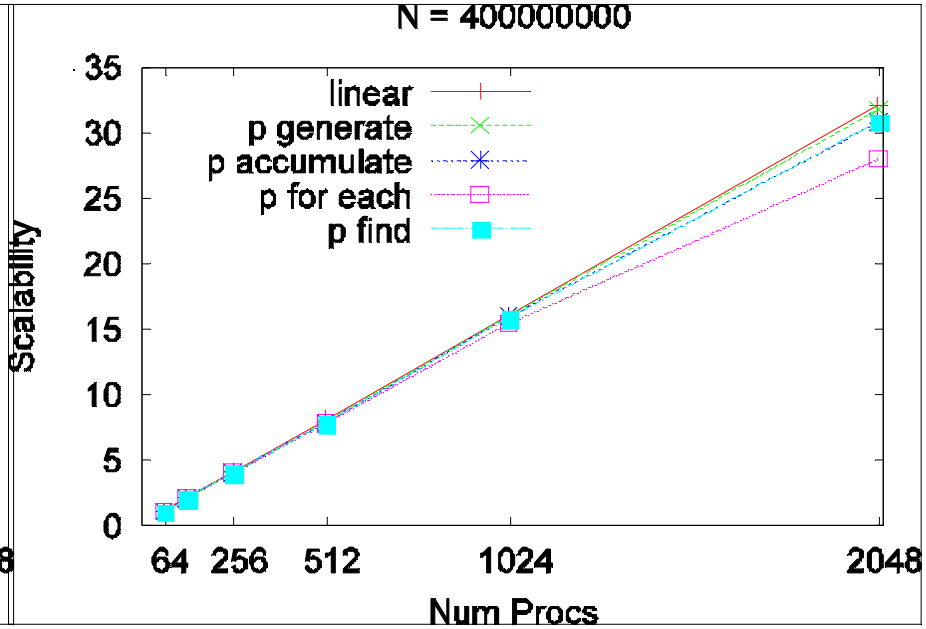
STAPL




```
pvector<double,default_distribution> x(N);
// …
double d = accumulate(default_view(x),0);
```

Scalability of pAlgorithms



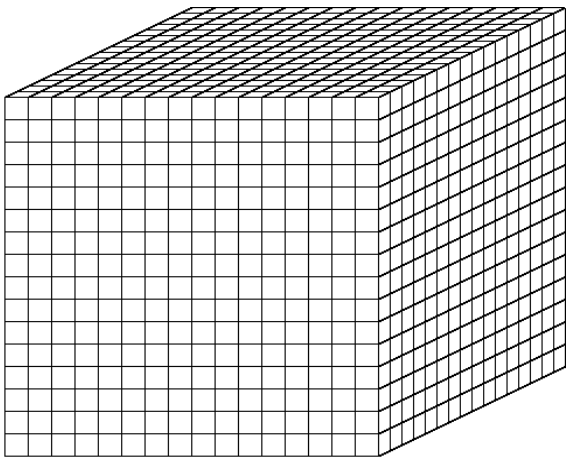


- Results obtained on an IBM P3 machine at NERSC
- Scalability is relative to 64 processors

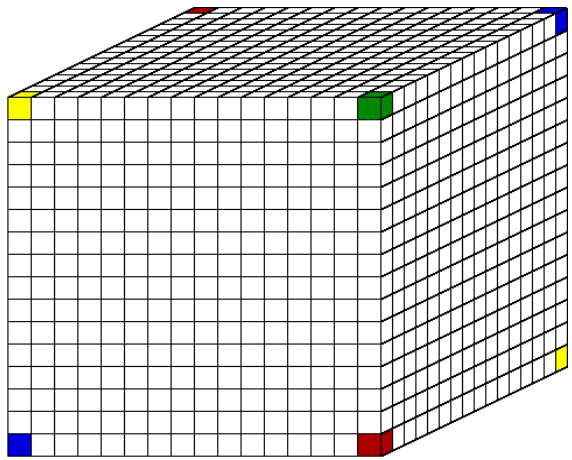
Discrete Ordinates Particle Transport Computation

Parasol
Smarter computing.
Texas A&M University

- Important application for DOE
  - E.g., Sweep3D and UMT2K
- Large, on-going DOE project at TAMU to develop application in STAPL (TAXI)



One sweep



Eight simultaneous sweeps

Parasol
Smarter computing.
Texas A&M University

Related work

Features\Project	STAPL	Charm++	PSTL	HTA	POOMA	Titanium	TBB
Language/Library	Lib	Lang	Lib	Lib	Lib	Lang	Lib
Memory Address Space	Shared	Shared/ Part	Shared	Shared	Shared	Shared/ Part	Shared
Programming Model	SPMD/ MPMD	MPMD	SPMD	SPMD	SPMD	SPMD/ MPMD	MPMD
Generic Data Type/ Generic Algorithms	Y/Y	Y/N	Y/Y	Y/N	Y/N	Y/Y	Y/Y
Reuse Seq Containers	Y	N	Y	N	N	Y	Y
Framework for pContainers	Y	N	N	N	N	N	Y
Data Structures (Array, Vector,List, Graph, Matrix)	A,V,L,G, M	V	V,L	M	A	A	V,H,Q
Views	Y	N	N	Y	N	Y	N
Data Partition/ Mapping	Y/Y	Y	N	Y	Y	N	N
Adaptive	Y	Y	N	N	N	N	N

So, what is “clean”?

Parasol
Smarter computing.
Texas A&M University

- == “not dirty”
  - Absence of extraneous matter
    - Remember Fortran!
    - Threads, locks, MPI, “annotations” are today’s assembler
  - Portability
    - Of correctness
    - Of performance

So, what's "clean"

- This isn't

```
void f(double* p, int n)
{
    if (n>GRAIN) {
      int m = n%GRAIN;
      //%% parallel
      for (int i = 0; i<min(m,N_PROCESSORS); ++i) {
                double* pp = p+i*GRAIN;
                acquire(lock[i]);
                // …
                release(lock[i]);
      }
      join();
}
```

So, what's "clean"?

Parasol
Smarter computing.
Texas A&M University

- Explicit locking
  - Instead: parallel algorithms, tasks, futures, message queues, parallel algorithms
- Explicit release of resource (e.g. lock)
  - RTTI
- Explicit calculation based of memory size or number of processors
  - Instead: supportive runtime
- Explicit tread management
  - Some notion of task

So, what's "clean"?

Parasol
Smarter computing.
Texas A&M University

- But everyone knows that!
  - No: "nobody" knows that
    - 99.9% of programmers have an extraordinarily naïve view of concurrency (threads and locks, if that)
    - And they will try to write concurrent code
- I'm sad that I can't teach the basics of concurrent programming to my freshmen
  - Worried really
- Example:

```
Array<double,3> m(xm,ym,zm);
// …
Array<double,2> v(zm) = 0;
parellel::for_each(range(0,zm),
                   <&>(i;v,m) { v[i] = parallel::accumulate(m[i]); }
```

Almost clean

clean

Who cares about “clean code”?

- Anybody
  - Who maintains code
  - Who is a beginner
  - Who is not an academic
  - Who uses “ordinary commercial software”

So

- Devote major efforts to
  - usability by non-expert programmers
    - Including computer scientists
  - usability by domain experts
- Much of the needed work is design more than academic research
  - Some is very serious research
  - Surprise: you can't completely separate the two
- Please