Our Approach: STAPL

- **STAPL**: Parallel components library
  - Extensible, open ended
  - Parallel superset of **STL**
  - Sequential inter-operability
  - Inter-operability with other libraries (Linpack, Trilinos)

- Layered architecture: User – Developer - Specialist
  - Extensible
  - Portable (only lowest layer needs to be specialized)

- **High Productivity Environment**
  - components have (almost) sequential interfaces.
STAPL Specification

- **STL Philosophy**
- **Shared Object View**
  - User Layer: No explicit communication
  - Machine Layer: Architecture dependent code
- **Distributed Objects**
  - no replication
  - no software coherence
- **Portable efficiency**
  - Runtime System virtualizes underlying architecture.
- **Concurrency & Communication Layer**
  - SPMD (for now) parallelism
The STAPL Programming Environment

User Code

pAlgorithms  pRange  pContainers

RTS + ARMI

Pthreads  OpenMP  MPI  Native

Native System
STAPL Overview

- Data is stored in pContainers
  - Parallel equivalents of all STL containers & more (e.g., pGraph)
- STAPL provides generic pAlgorithms
  - Parallel equivalents of STL algorithms & more (e.g., list ranking)
- pRanges bind pAlgorithms to pContainers
  - Similar to STL iterators, but also support parallelism
STAPL Overview

- pContainers
- pRange
- pAlgorithms
- RTS & ARMI Communication Infrastructure
- Applications using STAPL
pContainer Overview

**pContainer**: A distributed (no replication) data structure with parallel (thread-safe) methods

- **Ease of Use**
  - Shared Object View
  - Handles data distribution and remote data access internally (no explicit communication)

- **Efficiency**
  - De-centralized distribution management
  - OO design to optimize specific containers
  - Minimum overhead over STL containers

- **Extendability**
  - A set of base classes with basic functionality
  - New pContainers can be derived from Base classes with extended and optimized functionality
pContainer provides different views for users with different needs/levels of expertise

- **Basic User view:**
  - a single address space
  - interfaces similar to STL containers

- **Advanced User view:**
  - access to data distribution info to optimize methods
  - can provide customized distributions that exploit knowledge of application
pContainer Design

– Base Sequential Container
  – STL Containers used to store data
– Distribution Manager
  – provides shared object view
– BasePContainer
pContainer Major Components: Base Sequential Container

- Developer must implement operations in Base Sequential Container Interface

```cpp
class BaseSequentialContainer {
    virtual void AddElement(const Data&, GID) = 0;
    virtual const Data& GetElement(GID) const = 0;
    virtual void SetElement(GID, const Data&) = 0;
    virtual void DeleteElement(GID) = 0;
    virtual bool ContainElement(GID) const = 0;
};
```
pContainer major Components: Distribution Manager

- Distribution Manager
  - provides shared object view
  - Functionality in Base:
    - local/remote tests
    - Methods to monitor Distribution (balance, locality, ...)
  - Methods to re-Distribute
  - All aspects Customizable
template< class Container_Part, class Distribution_Manager >

class BasePContainer {

    //major attributes
    Distribution_Manager distribution;
    vector<STL_Containers> stl_container_collection;

    //major methods providing Shared Object View
    virtual void AddElement(Data);
    virtual Data GetElement(GID);
    virtual void SetElement(GID, Data);
    virtual void DeleteElement(GID);
    virtual bool IsLocal(GID);
    virtual Location LookUp(GID);  
}
STAPL Overview

- pContainers
- pRange
- pAlgorithms
- RTS & ARMI Communication Infrastructure
- Applications using STAPL
**pRange Overview**

- Interface between pAlgorithms and pContainers
  - pAlgorithms expressed in terms of pRanges
  - pContainers provide pRanges
  - Similar to STL Iterator
- Parallel programming support
  - Expression of computation as parallel task graph
  - Stores DDGs used in processing subranges
- Less abstract than STL iterator
  - Access to pContainer methods
- Expresses the Data—Task Parallelism Duality
pRange

- View of a work space
  - Set of tasks in a parallel computation
- Can be recursively partitioned into subranges
  - Defined on disjoint portions of the work space
  - Leaf subrange in the hierarchy
    - Represents a single task
    - Smallest schedulable entity

- Task:
  - Function object to apply
    - Using same function for all subranges results in SPMD
  - Description of the data to which function is applied
**pRange Example**

- Each subrange is a task
- Boundary of each subrange is a set of cut edges
- Data from several threads in subrange
  - If pRange partition matches data distribution then data access is all local
pRange Example

- Each subrange has a boundary and a function object
- Data from several threads in subrange
  - pMatrix is distributed
  - If subrange partition matches data distribution then all data access is local
- DDGs can be defined on subranges of the pRange and on elements inside each subrange
  - No DDG is shown here
- Partitioning of subrange
  - Subranges can be recursively partitioned
  - Each subrange has a function object
- Subranges of pRange
  - Matrix elements in several subranges
  - Each subrange has a function object
Overview

- pContainers
- pRange
- pAlgorithms
- RTS & ARMI Communication Infrastructure
- Applications using STAPL
pAlgorithms

- **pAlgorithm** is a set of parallel task objects
  - input for parallel tasks specified by the pRange
  - (Intermediate) results stored in pContainers
  - ARMI for communication between parallel tasks

- **pAlgorithms in STAPL**
  - Parallel counterparts of STL algorithms provided in STAPL
  - STAPL contains additional parallel algorithms
    - List ranking
    - Parallel Strongly Connected Components
    - Parallel Euler Tour
    - etc
### STAPL Counterparts to STL

<table>
<thead>
<tr>
<th></th>
<th>STL</th>
<th>STAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Storage</strong></td>
<td>Containers</td>
<td>pContainers</td>
</tr>
<tr>
<td></td>
<td>(e.g. vector, list, set)</td>
<td>(e.g. pVector, pList, pSet, pGraph, pArray, etc.)</td>
</tr>
<tr>
<td><strong>Data Access</strong></td>
<td>Iterators, Container ops</td>
<td>pRange, pContainer ops</td>
</tr>
<tr>
<td><strong>Data Distribution</strong></td>
<td>N/A</td>
<td>pContainers</td>
</tr>
<tr>
<td><strong>Algorithms Provided</strong></td>
<td>Defined by Standard</td>
<td>Parallel STL Equivalents, Graph Algorithms (e.g. pSCC, Parallel Euler Tour), List Ranking, pMatrix Algorithms</td>
</tr>
<tr>
<td><strong>Parallel Execution &amp; Management</strong></td>
<td>N/A</td>
<td>Executor, Scheduler, ARMI DDGs stored in pRange</td>
</tr>
<tr>
<td><strong>End-user Code Written Using</strong></td>
<td>Algorithms, Iterators, Containers</td>
<td>pAlgorithms, pRanges, pContainers</td>
</tr>
</tbody>
</table>

#### STL Code

```cpp
vector<int> v;
... initialization of 'v' ...
sort( v.begin(), v.end() );
```

#### STAPL Code

```cpp
pVector<int> pv;
... initialization of 'pv' ...
pSort( pv.get_pRange() );
```
Overview

- pContainers
- pRange
- pAlgorithms
- RTS & ARMI Communication Infrastructure
- Applications using STAPL
Current Implementation Protocols

- **Shared-Memory** (OpenMP/Pthreads)
  - shared request queues

- **Message Passing** (MPI-1.1)
  - sends/receives

- **Mixed-Mode**
  - combination of MPI with threads
  - flat view of parallelism (for now)
    - take advantage of shared-memory
STAPL Run-Time System

- **Scheduler**
  - Determine an execution order (DDG)
  - Policies:
    - Automatic: Static, Block, Dynamic, Partial Self Scheduling
    - User defined

- **Executor**
  - Execute DDG
    - Processor assignment
    - Synchronization and Communication
ARMI: STAPL Communication Infrastructure

ARMI: Adaptive Remote Method Invocation
- abstraction of shared-memory and message passing communication layer
- programmer expresses fine-grain parallelism that ARMI adaptively coarsens
- support for sync, async, point-to-point and group communication

ARMI can be as easy/natural as shared memory and as efficient as message passing
ARMI Communication Primitives

`armi_async`

- statement: tell a thread something
- non-blocking: doesn’t wait for request arrival or completion method invocation

```cpp
template<class Class, class Rtn, class Arg1...>
void armi_async(int destThread, armiHandle handle,
                 Rtn (*method)(Arg1...), Arg1 a1... )
```
ARMI Communication Primitives

- **armi_sync**
  - question: ask a thread something
  - blocking version
    - function doesn’t return until answer received from rmi
  - non-blocking version
    - function returns without answer
    - program can poll with rtnHandle.ready() and then access armi’s return value with rtnHandle.value()

- **collective operations**
  - armi_broadcast, armi_reduce, etc.
  - can adaptively set groups for communication
  - arguments always passed by value
ARMI Synchronization Primitives

- **armi_fence, armi_barrier**
  - tree-based barrier
  - implements distributed termination algorithm to ensure that all outstanding ARMI requests have been sent, received, and serviced
- **armi_wait**
  - blocks until at least one (possibly more) ARMI request is received and serviced
- **armi_flush**
  - empties local send buffer, pushing outstanding ARMI requests to remote destinations