STAPL: A High Productivity Programming Infrastructure for Parallel and Distributed Computing

Nancy Amato & Lawrence Rauchwerger

http://parasol.tamu.edu/stapl/

Parasol Lab, Dept of Computer Science, Texas A&M
Motivation

- There is a growing need for parallel programs
  - Large scale parallel machines getting larger
  - small scale parallel machines (i.e., multicores) are becoming ubiquitous
- Challenges
  - Parallel programming is specialized & costly
  - portability
  - Scalability & Efficiency is (usually) poor
  - Composability and integration with other components
STAPL

- **STAPL**: Parallel components library and development environment
  - Extensible & composable
  - Parallel superset of **STL**
  - Inter-operable with STL
- Layered architecture: User – Developer - Specialist
  - Extensible
  - Portable (only lowest layer needs to be specialized once at system installation)
- High Productivity Environment
  - Components have (almost) sequential interfaces
Applications Using STAPL

- Nuclear Eng. - Discrete Ordinates Particle Transport
- Bioinformatics - Protein Folding
- Geophysics - Seismic Ray Tracing
- Aerospace - Lattice Boltzmann Method
Outline

- Motivation

- STAPL: Standard Template Adaptive Parallel Library
  - Philosophy & Design Overview
  - Components for Program Development
    - pContainers, Views, pRange, pAlgorithms
  - Portability and Optimization
    - RTS & ARMI Communication Library
    - Adaptive Components, Adaptive Algorithm Selection
  - Applications developed using STAPL

- Summary & Future Work
STAPL Specification

- Adopts STL Philosophy: Generic Programming
  - composable (unlike STL)
  - Interoperable with other libraries and packages
- Shared Object View
  - User Layer: No explicit communication
  - Machine Layer: Architecture dependent code
- Distributed Objects
  - no replication (visible to the user)
  - no software coherence (visible to the user)
- Portable efficiency
  - Adaptation to environment
  - Runtime System virtualizes underlying architecture
- Compiles with any standard C++ compiler (GCC) (It’s a library – not a language)
**STL: Standard Template Library**

Generic programming components using C++ templates.

- **Containers - collection of other objects**
  - vector, list, deque, set, multiset, map, multi_map, hash_map
  - Templated by data type: `vector<int> v(50);`

- **Algorithms - manipulate the data stored in containers**
  - count(), reverse(), sort(), accumulate(), for_each(), reverse()

- **Iterators - Decouple algorithms from containers**
  - Provide generic *elementary access* to data in containers
  - can define custom *traversal* of container (e.g., every other element)
  - `count(vector.begin(), vector.end(), 18);`
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)
Usage Model

- Two Models:

<table>
<thead>
<tr>
<th>Application Programmer</th>
<th>Library Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single threaded</td>
<td>Multithreaded</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>PGAS</td>
</tr>
<tr>
<td>Implicit Synchronizations</td>
<td>Explicit Communications</td>
</tr>
<tr>
<td>Thread Safe Operations</td>
<td></td>
</tr>
</tbody>
</table>

- Data and Task Parallelism co-exist
- programmers can use library developer options, if desired
Outline

• Motivation

• STAPL: Standard Template Adaptive Parallel Library
  – Philosophy & Design Overview
  – Components for Program Development
    ▪ pContainers, Views, pRange, pAlgorithms
  – Portability and Optimization
    ▪ RTS & ARMI Communication Library
    ▪ Adaptive Components, Adaptive Algorithm Selection
  – Applications developed using STAPL

• Summary & Future Work
pContainers

Generic, distributed data structures with parallel methods

- **Usability**
  - Shared object view
  - Generic access mechanism through Views
  - Handle data distribution and remote data access internally
  - Interface compatible with sequential counterpart

- **Extendability & Composability**
  - New (user defined) pContainers extend Base classes
  - pContainers of pContainers

- **Efficiency & Adaptability**
  - OO design to optimize specific containers
  - Optional user customization of each pContainer instance
    template parameters & traits
    - Enable/Disable Performance Monitoring, thread safety, ...
    - Select Partition Strategies, consistency models, ...

- pContainers currently available in STAPL
  - pVector, pList, pMap, pSet, pMultiMap, pMultiSet, pHashMap, pHashSet, pArray, pGraph, pMatrix
Example: pArray

- pArray is an ordered sequence of elements accessed using indices.
- The domain (unique identifiers of pContainer elements) is a range of integers, e.g., [0,7].
- STAPL provides common
  - Logical Partitions of elements (e.g., Blocked, Balanced, BlockCyclic, …)
  - PartitionMappers to map subdomains to locations (e.g., Blocked, Cyclic, …)
  - Users can extend existing & implement new Partitions and Mappers.

Ex 1: Blocked Partition
Cyclic Mapper

Ex 2: Cyclic Partition
Blocked Mapper
Views

- Views provide generic access mechanism for pContainer
  - STAPL equivalent of STL iterator, extended to allow for efficient parallelism
  - Focus on processing range of items, instead of single item
  - Specify different partitions and data traversal for the data in a pContainer
  - Hierarchically defined to control locality and granularity of parallelism

Data Space

Two logical views - one (left) aligned with physical partition
pAlgorithms

- **pAlgorithms in STAPL**
  - Parallel counterparts of STL algorithms
  - Additional parallel algorithms
    - Common parallel algorithms:
      - Prefix sums
      - List ranking
    - pContainer specific algorithms:
      - Strongly Connected Components (pGraph)
      - Euler Tour (pGraph)
      - Matrix multiplication (pMatrix)
  - Composable
    - pAlgorithms may invoke pAlgorithms
  - multiple algorithms exist for a particular operation (e.g., sorting) and STAPL adaptively selects which to use
pAlgorithms and Views

- Views are inputs for pAlgorithms
  - Input views specify parallelism the algorithm can exploit

- A pAlgorithm may operate on views before beginning to optimize the computation
  - Refining (subdividing) a sub-view may increase parallelism and locality
    - Views may be adjusted to match data boundary
    - Different views may be adjusted differently to allow easier algorithm specification

- Optimized access methods when all the data of a sub-view is local and contiguous
pRange - Task Graphs in STAPL

- **Task**
  - Work function
  - Data to process

- **Task dependencies**
  - Expressed in Task Dependence Graph (TDG)
  - TDG queried to find tasks ready for execution
pRange: Simple Dependence Specification

- **Goal**: Developer expresses dependencies concisely
  - If needed, full enumeration of dependencies is supported

- **Common patterns supported in pRange**
  - Sequential – sources depend on sinks
  - Independent – no new dependencies needed in composed graph
  - Pipelined – dependencies follow a regular pattern
    - Wave front, tree-based reductions, etc.
Scalability of pAlgorithms

- Results obtained on an IBM P3 machine at NERSC
  - Strong scaling: same problem size as increase #procs
    - Results show scalability relative to 64 processors
  - Weak scaling: increase problem size as increase #procs, keep work per processor constant
Outline

• Motivation

• STAPL: Standard Template Adaptive Parallel Library
  – Philosophy & Design Overview
  – Components for Program Development
    ▪ pContainers, Views, pRange, pAlgorithms
  – Portability and Optimization
    ▪ RTS & ARMI Communication Library
    ▪ Framework for Algorithm Selection and Tuning (FAST)

• Applications developed using STAPL

• Summary & Future Work
Run Time System

Smart Application

Application Specific Parameters

STAPL RTS

Advanced stage

- ARMI
- Executor
- Memory Manager

Experimental stage: multithreading

- ARMI
- Executor

Custom scheduling

- e.g., K42 User-Level Dispatcher
- Kernel scheduling

Operating System

Kernel Scheduler

(no custom scheduling, e.g., NPTL)
STAPL RTS

- Adaptive Remote Method Invocation (ARMI) Communication Library
  - Synchronous and Asynchronous RMI
  - Support for message aggregation
  - Synchronization primitives
    - Fence, Global Distributed Locks, Group-Based Synchronization
- Executor/Scheduler - execute pRange tasks
  - Customized task scheduling & load balancing for every pRange instance
  - RTS selects default policy, but can be user specified
- Performance monitor for feedback to the user and the adaptive framework
- Multithreaded RTS
  - RMI servers, Task executors, Comm. threads, ...
RTS Consistency Models

**Processor Consistency** *(default)*
- Accesses from a processor on another’s memory are sequential
- Requires in-order processing of RMIs
  - Limited parallelism

**Object Consistency**
- Accesses to different objects can happen out of order
- Uncovers fine-grained parallelism
  - Accesses to different objects are concurrent
  - Potential gain in scalability
- Can be made default for specific computational phases

**Mixed Consistency**
- Use Object Consistency on select objects
  - Selection of objects fit for this model can be:
    - Elective – the application can specify that an object’s state does not depend on others’ states.
    - Detected – if it is possible to assert the absence of such dependencies
- Use Processor Consistency on the rest
Outline

• Motivation

• STAPL: Standard Template Adaptive Parallel Library
  – Philosophy & Design Overview
  – Components for Program Development
    ▪ pContainers, Views, pRange, pAlgorithms
  – Portability and Optimization
    ▪ RTS & ARMI Communication Library
    ▪ Adaptive Components, Adaptive Algorithm Selection
  – Applications developed using STAPL

• Summary & Future Work
Support for Adaptivity in STAPL

- **pAlgorithms**
  - Algorithm Selection and Tuning (FAST)
  - Parametric Algorithms

- **pContainers**
  - Consistency model may vary for each instance, or even over time for the same instance
  - Data distribution can be selected/modified at runtime

- **ARMI Communication Library**
  - Message passing and/or shared memory communication modes
  - Aggregation of messages to tolerate latency
Adaptive Algorithm Selection (ICS 00, IWACT 01, PPoPP 05)

Overview of Approach

- **Given**
  Multiple algorithmic choices for the same high level operation

- **At STAPL installation (& refine later)**
  Analyze each pAlgorithm’s performance on system and create a selection model

- **Program execution**
  Gather parameters, query model, and select pAlgorithm to use
Parallel Sorting - Relative Performance

- Relative performance of parallel sorting algorithms on SGI Altix
- Adaptive algorithm selection model obtains 99.7% of the possible performance.
- Next best algorithm (sample sort) provides only 90.4%.
Parallel Sorting - performance penalty

- Relative performance penalty of parallel sorting algorithms on SGI Altix
- Adaptive algorithm selection model incurs 12% average penalty on misprediction
- Next best algorithm (sample sort) averages 132% penalty
Outline

• Motivation

• STAPL: Standard Template Adaptive Parallel Library
  – Philosophy & Design Overview
  – Components for Program Development
    ▪ pContainers, Views, pRange, pAlgorithms
  – Portability and Optimization
    ▪ Adaptive Remote Method Invocation (ARMI) Communication Library
    ▪ Adaptive Components, Adaptive Algorithm Selection
  – Applications developed using STAPL

• Summary & Future Work
Discrete Ordinates Particle Transport Computation

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

- Spatial sweep for each direction of particle transport, each represented by a pRange where the sweep is encoded in the pRange’s TDG (task dependence graph)
## Related work

<table>
<thead>
<tr>
<th>Features/Project</th>
<th>STAPL</th>
<th>Charm++</th>
<th>PSTL</th>
<th>HTA</th>
<th>POOMA</th>
<th>Titanium</th>
<th>TBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language/Library</td>
<td>Lib</td>
<td>Lang</td>
<td>Lib</td>
<td>Lib</td>
<td>Lib</td>
<td>Lang</td>
<td>Lib</td>
</tr>
<tr>
<td>Memory Address Space</td>
<td>Shared</td>
<td>Shared/Part</td>
<td>Shared</td>
<td>Shared</td>
<td>Shared</td>
<td>Shared/Part</td>
<td>Shared</td>
</tr>
<tr>
<td>Programming Model</td>
<td>SPMD/MPMD</td>
<td>MPMD</td>
<td>SPMD</td>
<td>SPMD</td>
<td>SPMD</td>
<td>SPMD/MPMD</td>
<td>MPMD</td>
</tr>
<tr>
<td>Generic Data Type/ Generic Algorithms</td>
<td>Y/Y</td>
<td>Y/N</td>
<td>Y/Y</td>
<td>Y/N</td>
<td>Y/N</td>
<td>Y/Y</td>
<td>Y/Y</td>
</tr>
<tr>
<td>Reuse Seq Containers</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Framework for pContainers</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Data Structures</td>
<td>A,V,L,G,M</td>
<td>V</td>
<td>V,L</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>V,H,Q</td>
</tr>
<tr>
<td>Views</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Data Partition/Mapping</td>
<td>Y/Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Adaptive</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Conclusion

- STAPL has been used successfully on several large-scale, complex applications
  - STAPL has influenced Intel’s TBB
- We plan to release a version soon to friendly users
- More info at http://parasol.tamu.edu/stapl/
- STAPL Team:
  - Faculty: Lawrence Rauchwerger (PI), Nancy Amato, Bjarne Stroustrup
  - Postdoc: Mauro Bianco
  - Students: Antal Buss, Olga Pearce, Antoniu Pop, Ioannis Papadopoulos, Timmie Smith, Gabriel Tanase, Nathan Thomas
  - Sponsors: DOE, NSF, IBM, Intel, HP