STAPL

Standard Template Adaptive Parallel Library

Lawrence Rauchwerger

Antal Buss, Harshvardhan, Ioannis Papadopoulous,
Olga Pearce, Timmie Smith, Gabriel Tanase, Nathan Thomas,
Xiabing Xu, Mauro Bianco, Nancy M. Amato

http://parasol.tamu.edu/stapl

Dept of Computer Science and Engineering, Texas A&M

Parasol
Smarter computing.
Texas A&M University

Motivation

- Multicore systems: ubiquitous
- Problem complexity and size is increasing
  - Dynamic programs are even harder
- Programmability needs to improve
- Portable performance is lacking
  - Parallel programs are not portable
  - Scalability & Efficiency is (usually) poor

STAPL: Standard Template Adaptive Parallel Library

Parasol

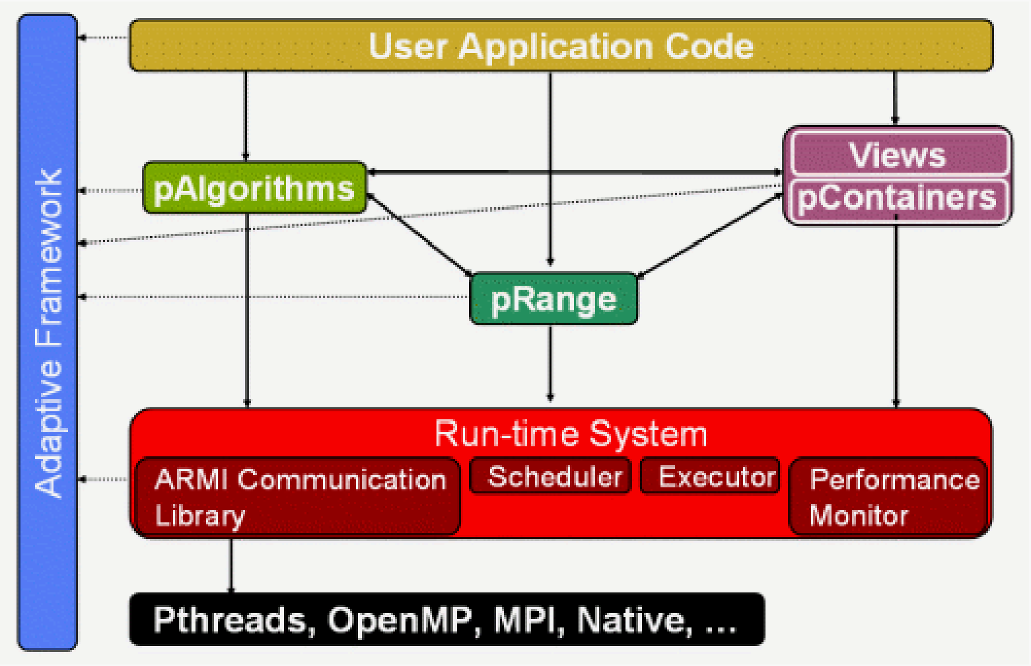
A library of parallel components that adopts the generic programming philosophy of the C++ Standard Template Library (STL)

–**Application Development Components**

- pAlgorithms, pContainers, Views, pRange
- Provide Shared Object View to eliminate explicit communication in application

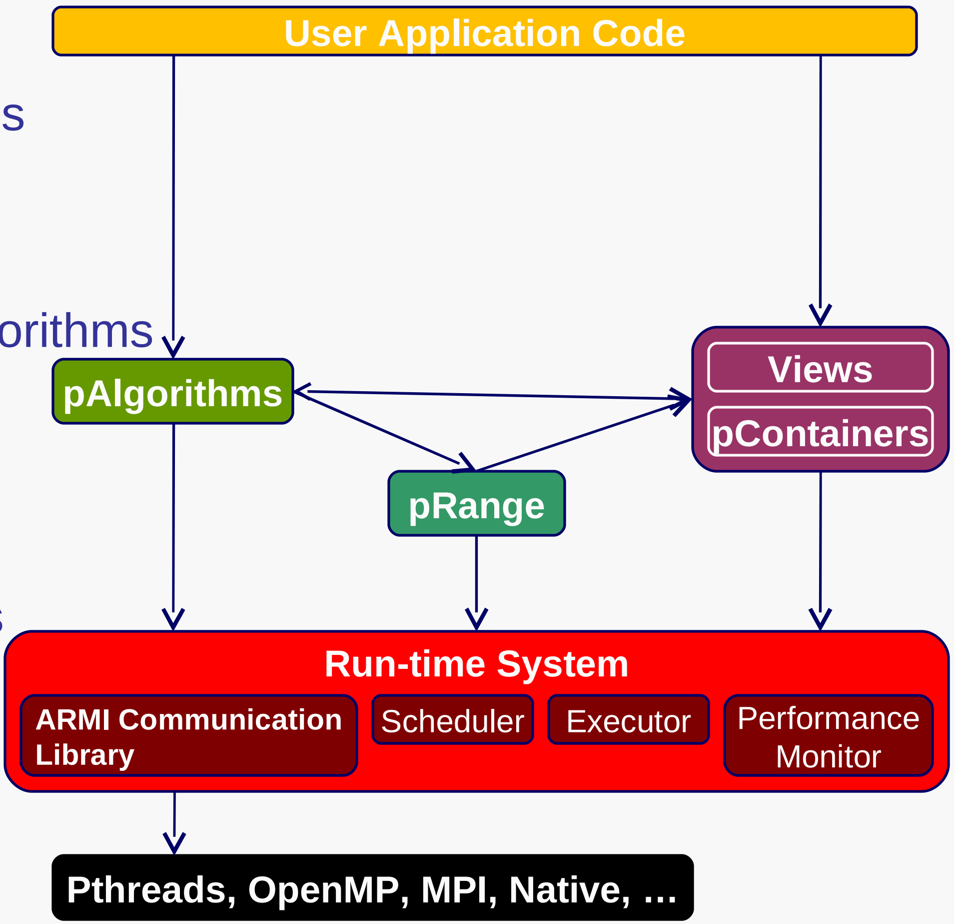
–**Portability and Optimization**

- Runtime System(RTS) and Adaptive Remote Method Invocation (ARMI) Communication Library
- Framework for Algorithm Selection and Tuning (FAST)



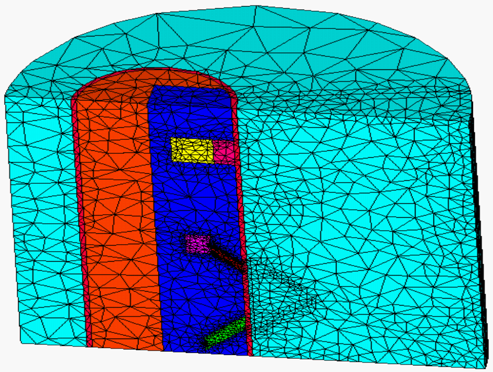
Three STAPL Developer Levels

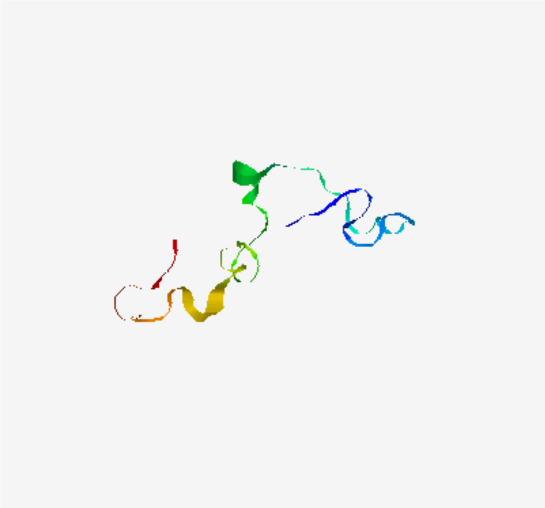
- Application Developer
  - Writes application
  - Uses pContainers and pAlgorithms
- Library Developer
  - Writes new pContainers and pAlgorithms
  - Uses pRange and RTS
- Run-time System Developer
  - Ports system to new architectures
  - Writes task scheduling modules
  - Uses native threading and communication libraries

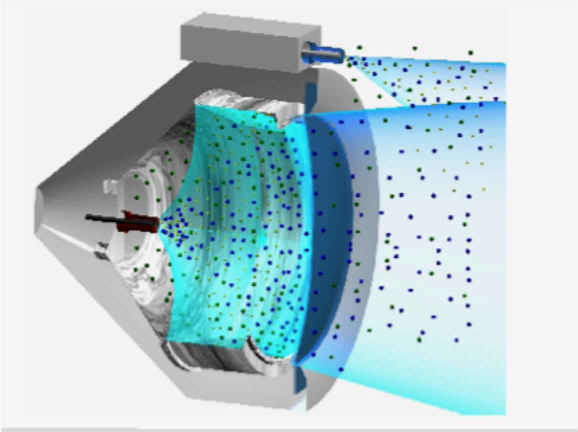


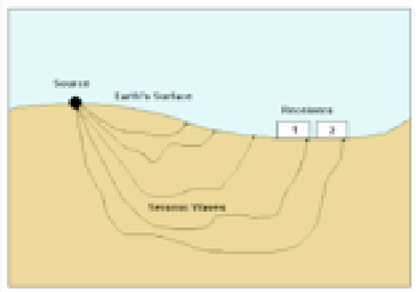
Applications Using STAPL

- Particle Transport - PDT
- Bioinformatics - Protein Folding
- Geophysics - Seismic Ray Tracing
- Aerospace - MHD
  - Seq. “Ctran” code (7K LOC)
  - STL (1.2K LOC)
  - STAPL (1.3K LOC)

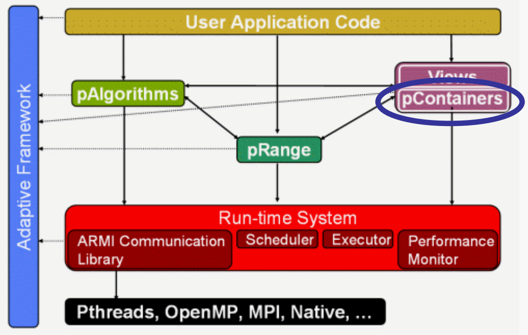








pContainers : Parallel Containers



- Container - Data structure with an interface to maintain and access a collection of generic elements
  - STL (vector, list, map, set, hash), MTL[1] (matrix), BGL[2] (graph), etc.

- pContainer - distributed storage and concurrent methods
  - Shared Object View
  - Compatible with sequential counterpart (e.g., STL)
  - Thread Safe
  - Support for user customization (e.g., data distributions)
  - Currently Implemented: pArray, pVector, pList, pGraph, pMatrix, pAssociative

1] Matrix Template Library 2] Boost Graph Library

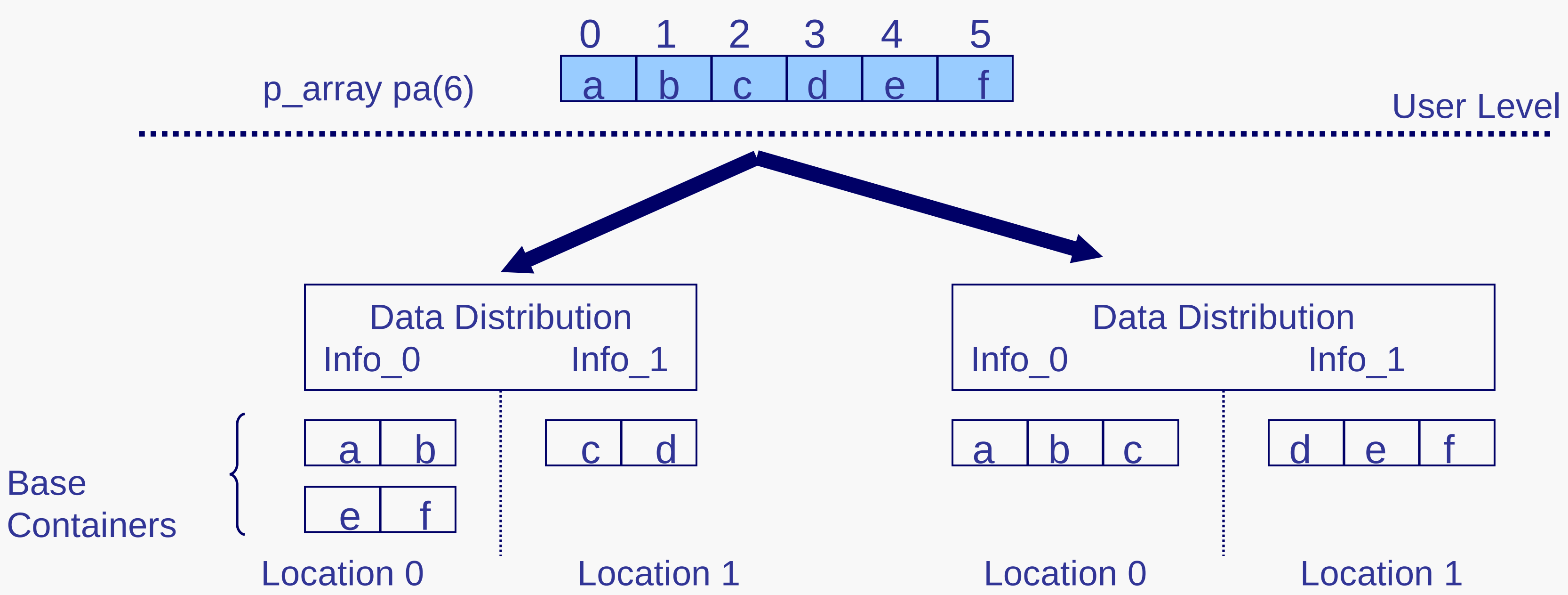
pContainer Framework

Concepts and methodology for developing parallel containers

- pContainers - a collection of base containers and information for parallelism management
- Improved user productivity
  - Base classes providing fundamental functionality
    - Inheritance
    - Specialization
  - Composition of existing pContainers
- Scalable performance
  - Distributed, non replicated data storage
  - Parallel (semi-random) access to data
  - Low overhead relative to the base container counterpart

pContainer Framework Concepts

- Base Container : data storage
  - sequential containers (e.g., STL, MTL, BGL)
  - parallel containers (e.g., Intel TBB)
- Data Distribution Information
  - Shared object view
  - Global Identifier, Domain, Partition, Location, Partition Mapper



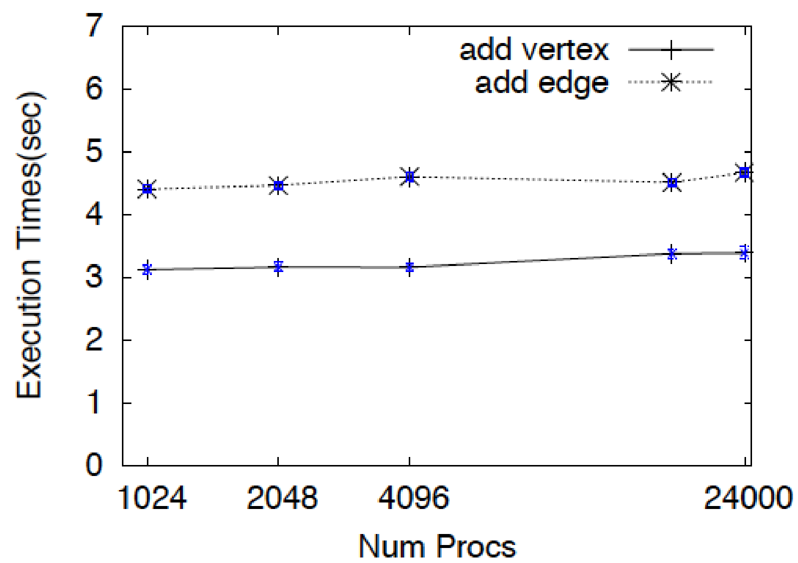
pContainer Interfaces

- Constructors
  - Default constructors
  - May specify a desired data distribution
- Concurrent Methods
  - Sync, async, split phase
- Views

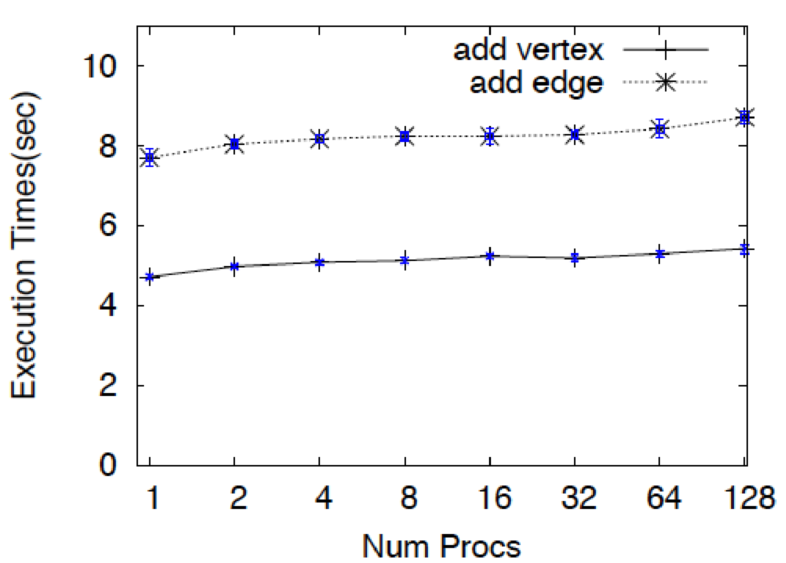
```
stapl_main(){
  partition_block_cyclic partition(10); //argument is block size
  p_matrix<int> data(100, 100, partition);
  p_generate(data.view(), rand());
  res=p_accumulate(data.view(),0);
}
```

pGraph Methods

- Performance for add vertex and add edge asynchronously
- Weak scaling on CRAY using up to 24000 cores and on Power 5 cluster using up to 128 cores
- Torus with 1500x1500 vertices per processor



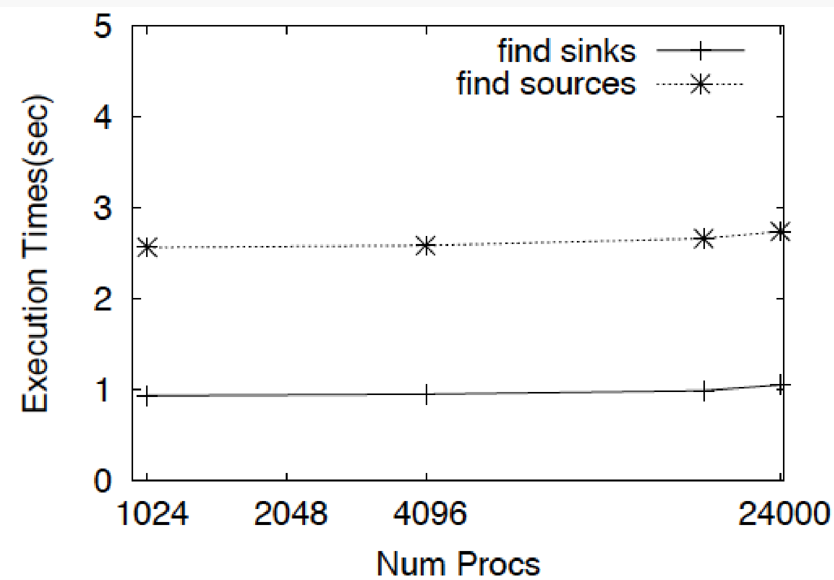
CRAY XT4



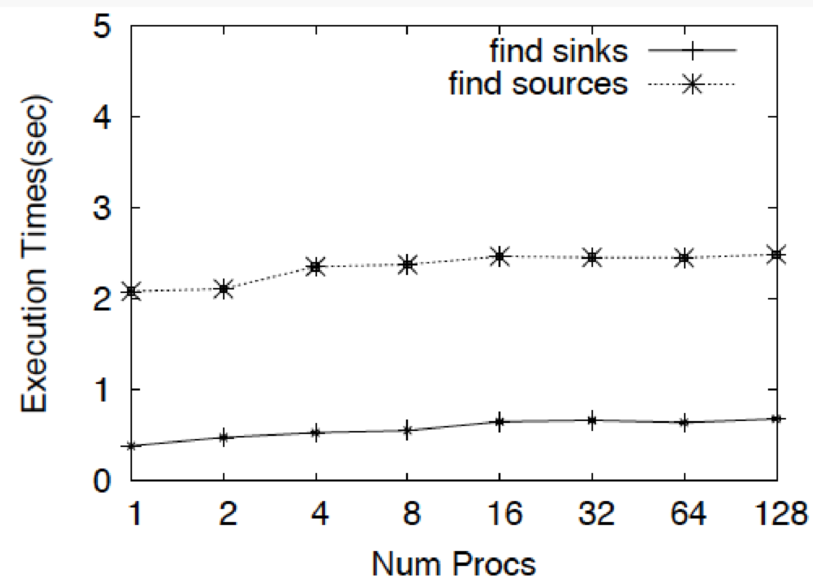
Power 5

pGraph Algorithms

- Performance for find_sources and find_sinks in a directed graph
- Weak scaling on CRAY using up to 24000 cores and on Power 5 cluster using up to 128 cores
- Torus with 1500x1500 vertices per processor

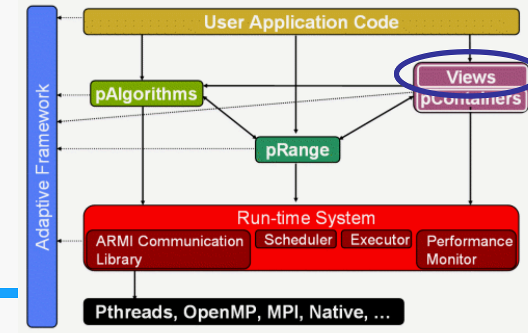


CRAY XT4

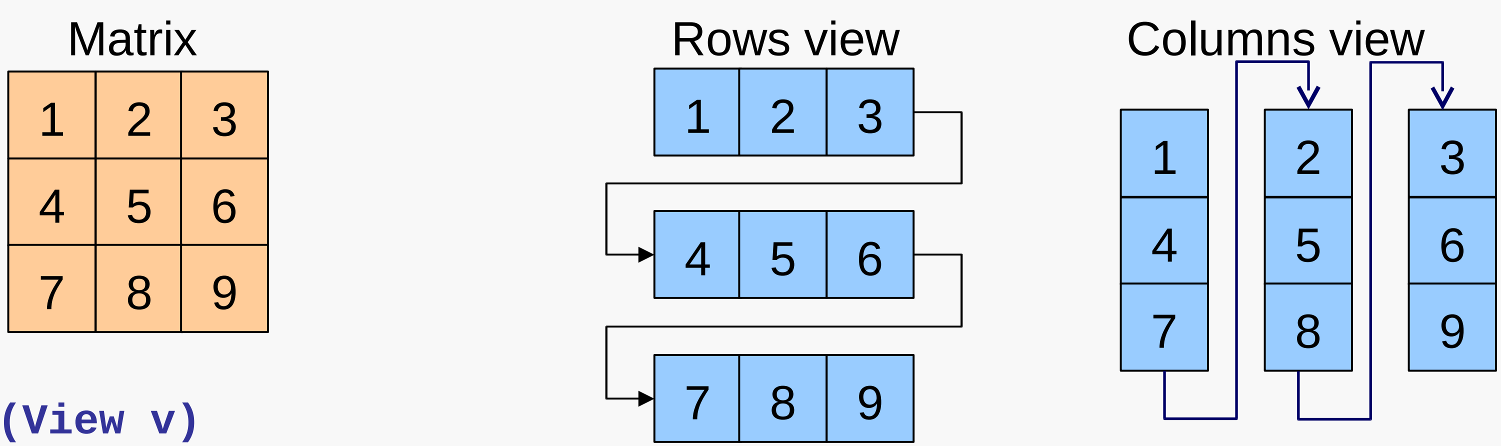


Power 5

Views



- A View defines an abstract data type that provides methods for access and traversal of the elements of a pContainer that is independent of how the elements are stored in the pContainer.
- Example: print the elements of a matrix

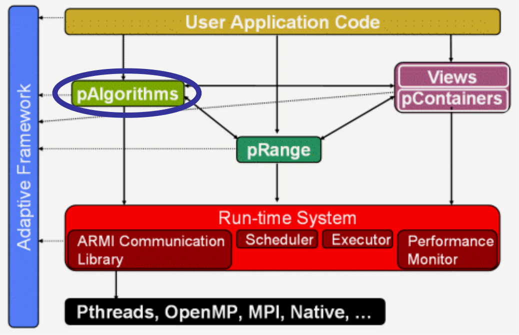



```
print(View v)
  for i=1 to v.size() do
    print(v[i])
```

Output
1,2,3,4,5,6,7,8,9

Output
1,4,7,2,5,8,3,6,9

pAlgorithms



- Build and execute task graphs to perform computation
  - Task graphs in STAPL are called pRanges
- Easy to develop
  - Work functions look like sequential code
  - Work functions can call STAPL pAlgorithms
  - pRange factories simplify task graph construction
- STAPL pAlgorithms accelerate application development
  - Basic building blocks for applications
  - Parallel equivalents of STL algorithms
  - Parallel algorithms for pContainers
    - Graph algorithms for pGraphs
    - Numeric algorithms/operations for pMatrices

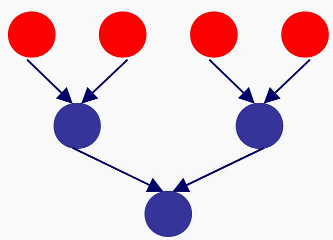
Parallel Find

- Find first element equal to the given value

```
View::iterator
p_find(View view, T value)
  return
  map_reduce(
    view,
    find_work_funtion(value),
    std::less()
  );
```

map operation

reduce operation



```
View::iterator
find_work_function(View view)
  if (do_nesting())
    return p_find(view, value)
  else
    return std::find(view.begin(),
                     view.end(),
                     value)
  endif
end
```

Parallel Sample Sort

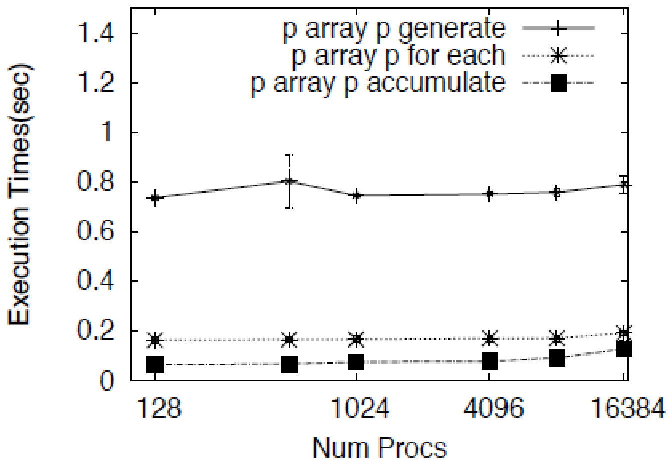
Parasol

- pAlgorithm written using sequence of task graphs.

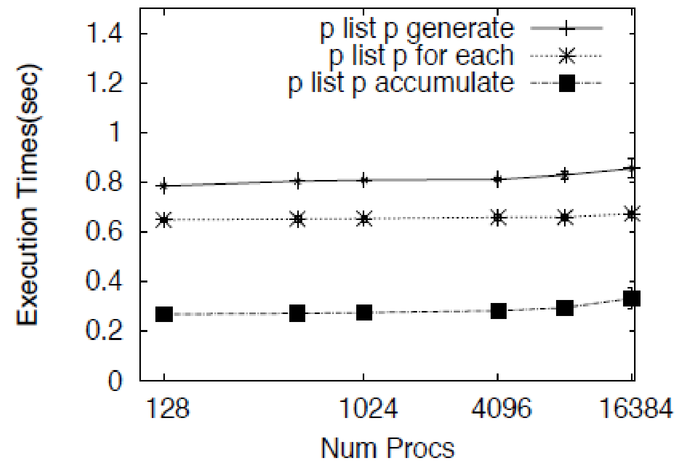
```
p_sort(View view, Op comparator)
  // handle recursive call
  if (view.size() <= get_num_locations())
    reduce(view, merge_sort_work_function(comparator));

  sample_view = map(view, select_samples_work_function());

  // sort the samples
  p_sort(sample_view, comparator);

  // paritition the data using the samples
  partitioned_view = map(view, full_overlap_view(sample_view),
                          bucket_partition_work_function(comparator));

  // sort each partition
  map(partitioned_view, sort_work_function(comparator));
```

Scalability of pAlgorithms

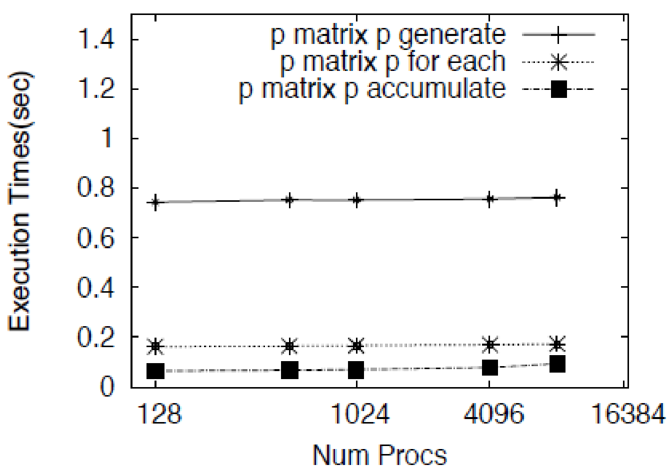
Execution times for weak scaling of pAlgorithms on data stored in different pContainers on CRAY XT4.



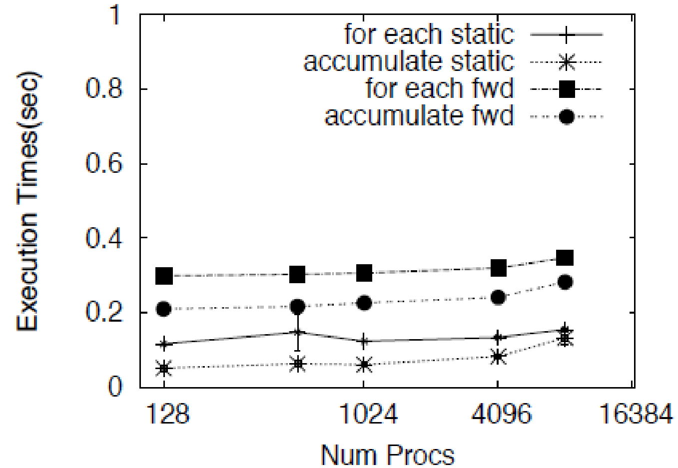
(a) **pArray**; 20M/Proc



(b) **pList**; 20M/Proc

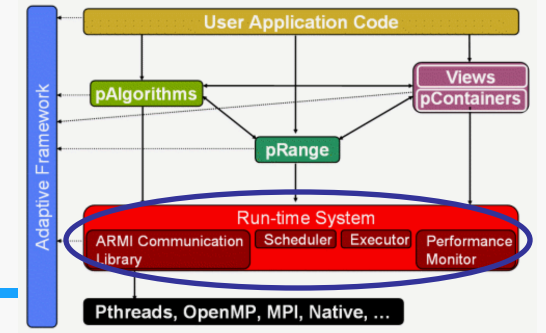


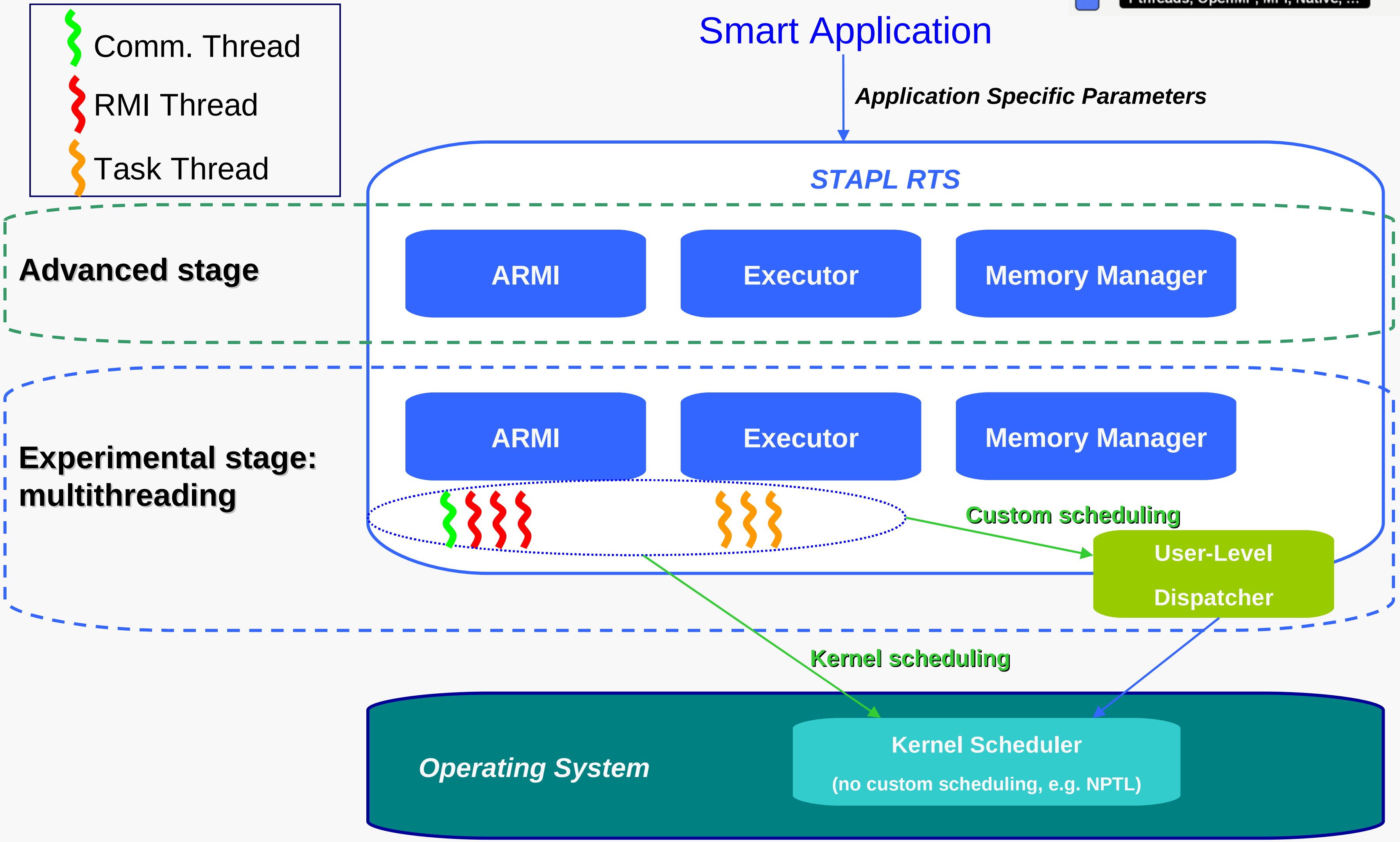
(c) **pMatrix**20M/Proc



(d) **pGraph** 1500x1500 stencil/Proc

STAPL Runtime System





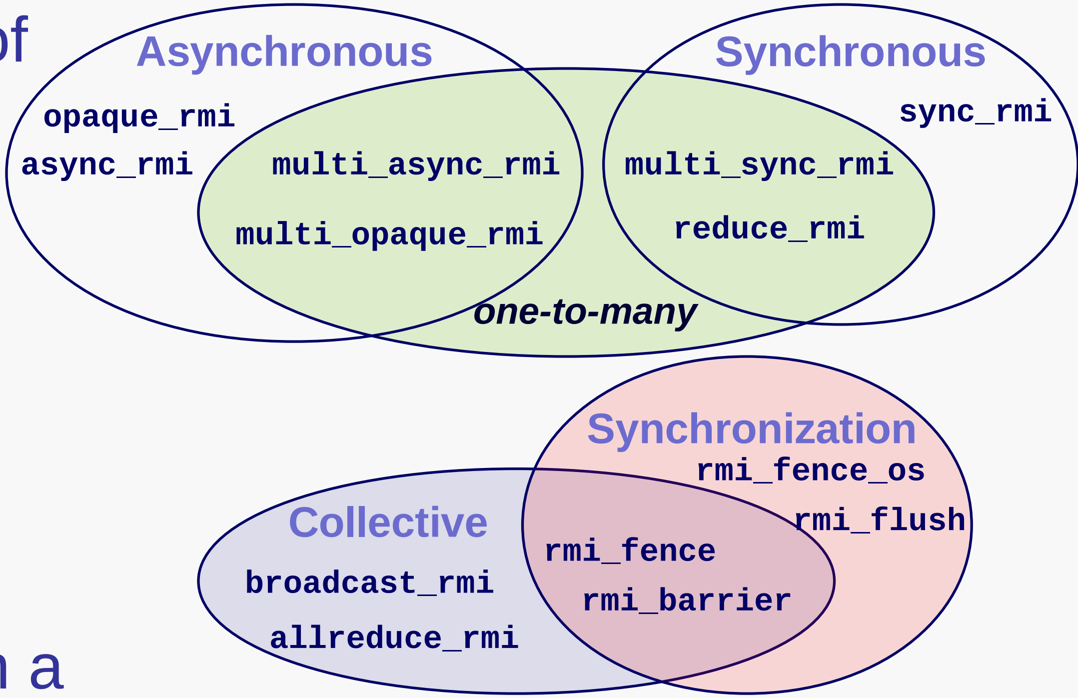
The STAPL Runtime System (RTS)...

- Abstracts platform resources
  - threads, mutexes, atomics
- Provides consistent API and behavior across platforms
- Configured at compile-time for a specific platform
  - Hardware counters, different interconnect characteristics
- Adapts at runtime at the runtime environment
  - Available memory, communication intensity etc.
  - Provides interface for calling functions on distributed objects
  - ARMI – Adaptive Remote Method Invocation
- There is one instance of the RTS running in every process
  - So it is distributed as well

ARMI:
Adaptive Remote Method Invocation

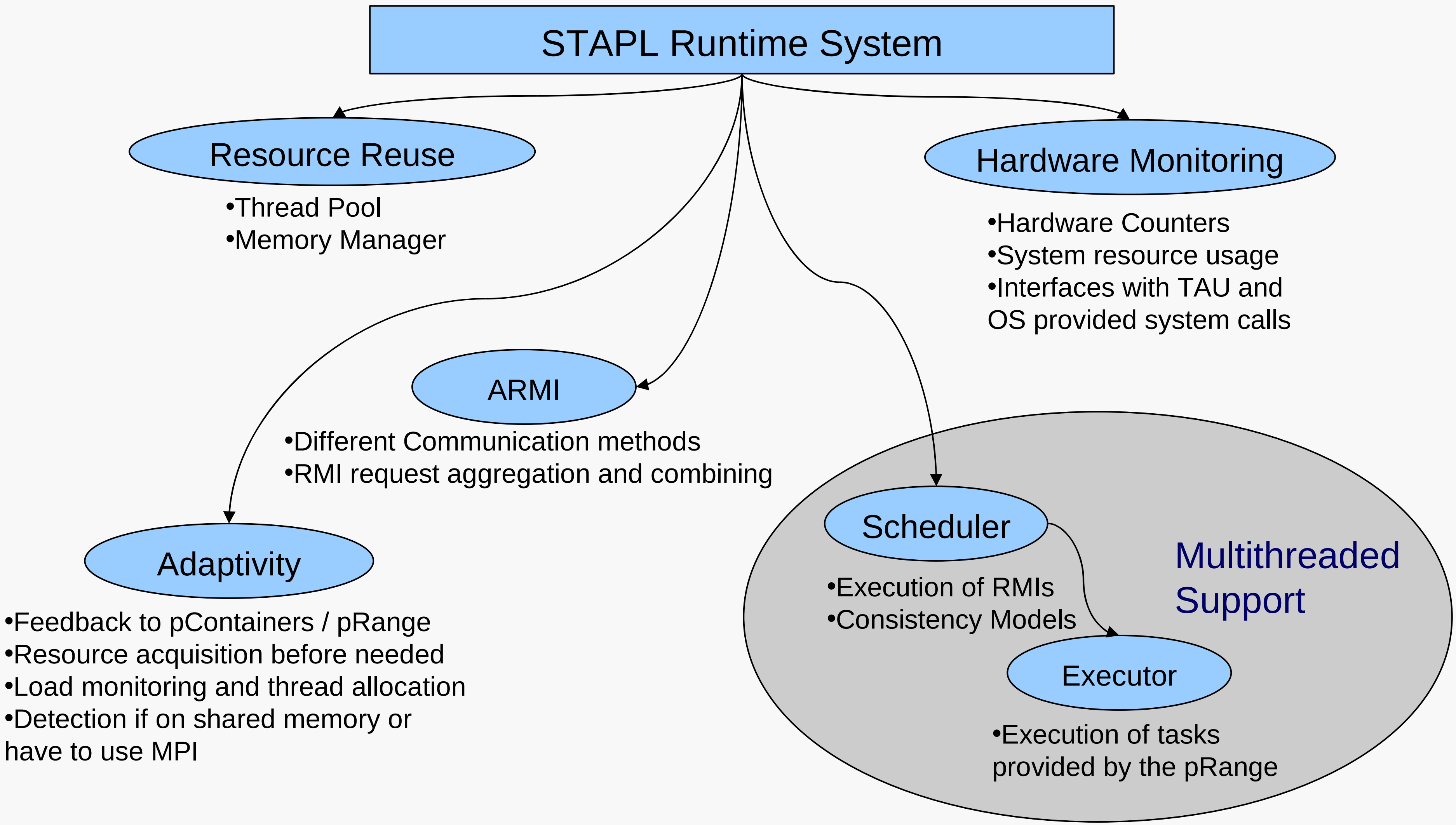
Parasol

- Communication service of the RTS
- Provides two degrees of freedom
  - Allows transfer of data, work, or both across the system
  - Used to hide latency
- Used to call a function on a distributed object anywhere on the system
- Supports a mixed-mode operation (MPI+threads)

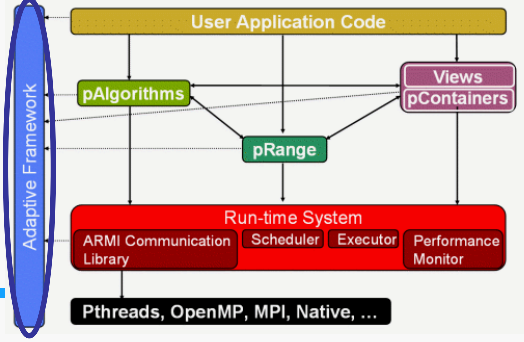



```
// Example of ARMI use
async_rmi(destination, p_object,
          function, arg0, ...);
r = sync_rmi(destination, p_object,
             function, arg0, ...);
```

The STAPL RTS: Major Components

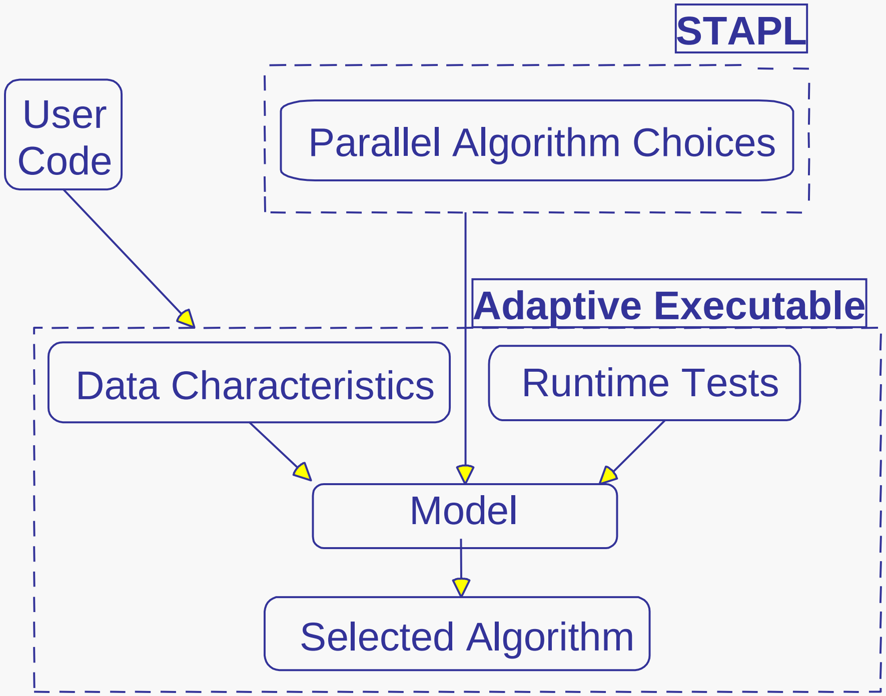


FAST Architecture



Parasol

- Framework for parallel algorithm selection
- Developer specifies important parameters, metrics to use
- Developer queries model produced to select implementation
- Selection process transparent to code calling algorithm



Parallel Sorting: Experimental Results

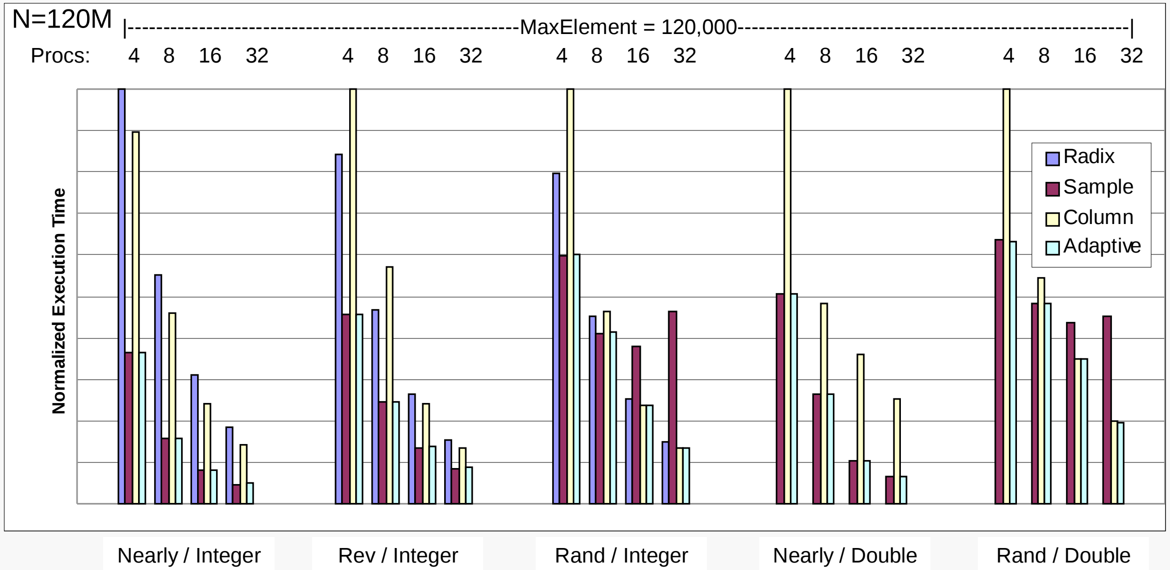
<u>Attributes for Selection Model</u>

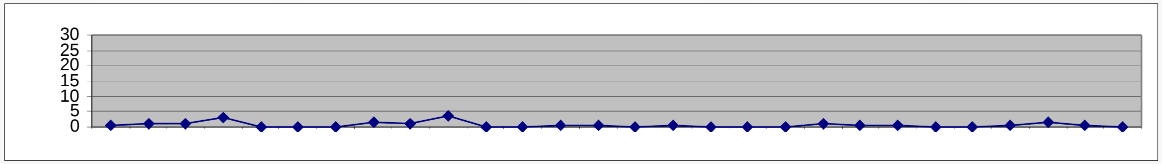
- –Processor Count
- –Data Type
- –Input Size
- –Max Value (impacts radix sort)
- –Presortedness

SGI Altix Selection Model

```
if p ≤ 8 then
  sort = "sample"
else
  if dist_norm ≤ 0.117188 then
    sort = "sample"
  else
    if dist_norm ≤ 0.370483 then
      sort = "column"
    else
      sort = "sample"
    end if
  end if
end if
```

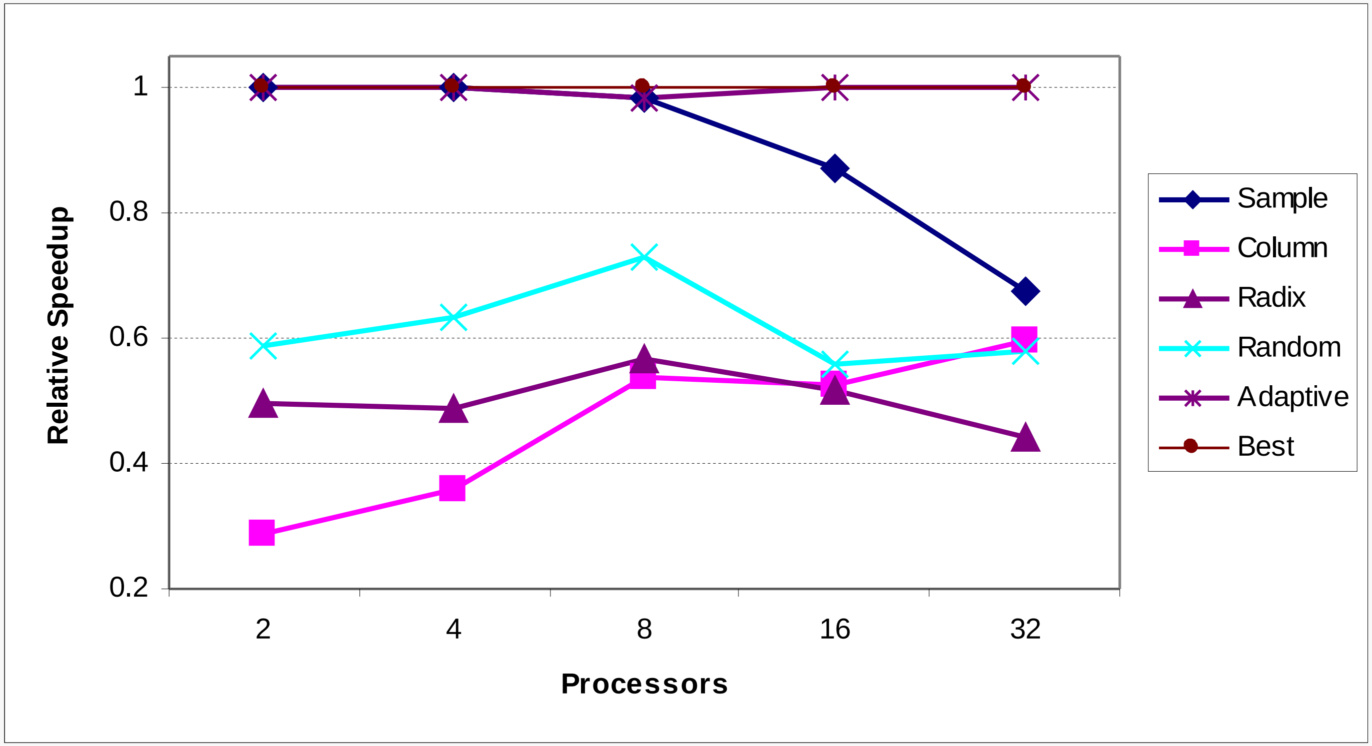
SGI Altix Validation Set (V1) – 100% Accuracy





Adaptive Performance Penalty

Parallel Sorting - Altix Relative Performance (V2)

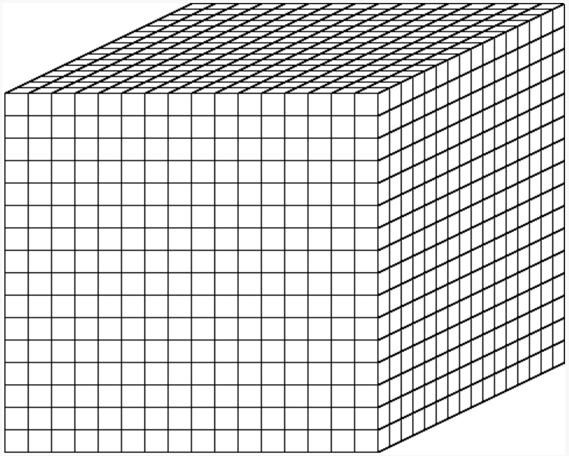


- Model obtains 99.7% of the possible performance.
- Next best algorithm (sample) provides only 90.4%.

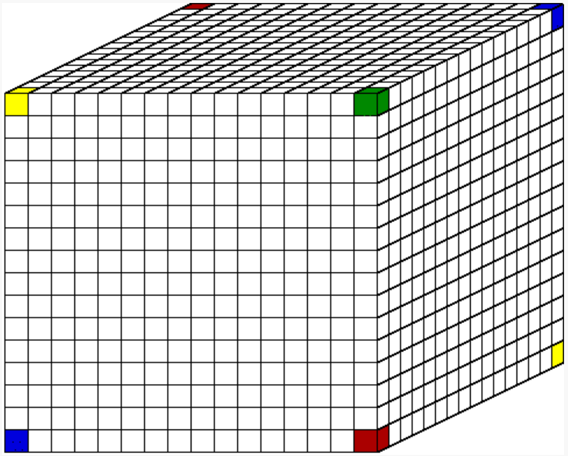
PDT:
Developing Applications with STAPL

Parasol

- Important application for DOE
  - E.g., Sweep3D and UMT2K
- Large, on-going DOE project at TAMU to develop application in STAPL
- STAPL precursor used by PDT in DOE PSAAP center

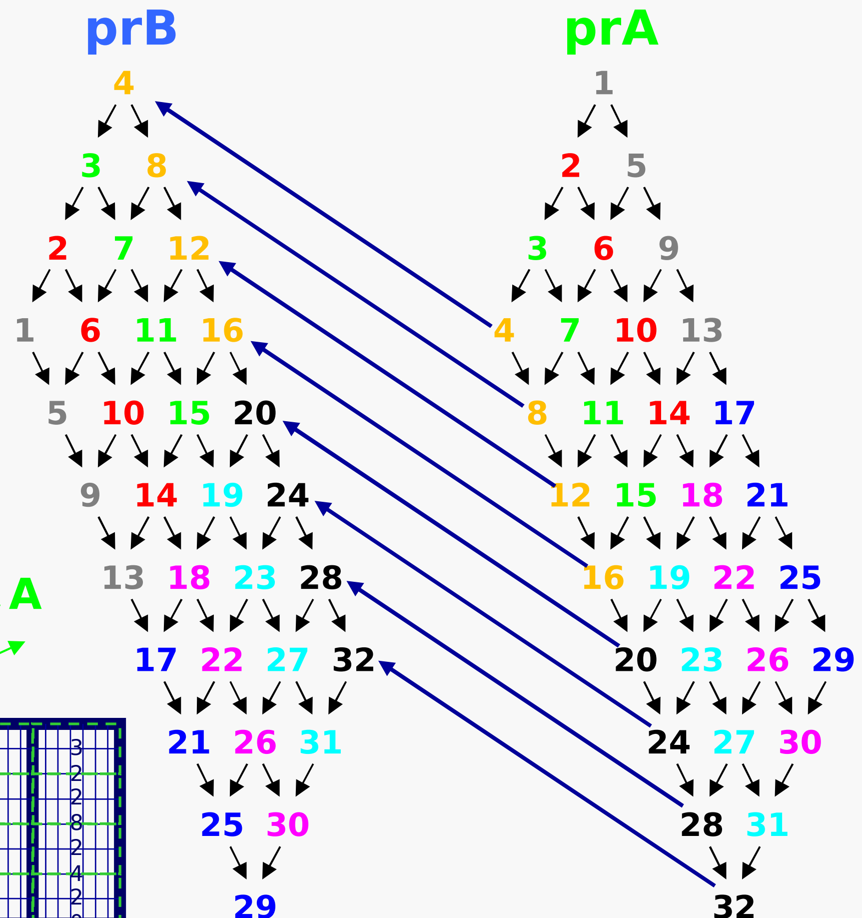


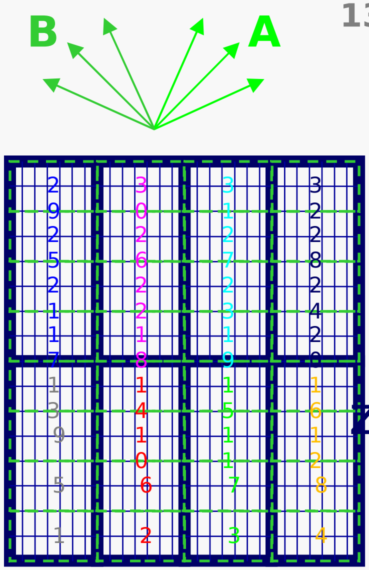
One sweep



Eight simultaneous sweeps

pRanges in PDT: Writing new pAlgorithms

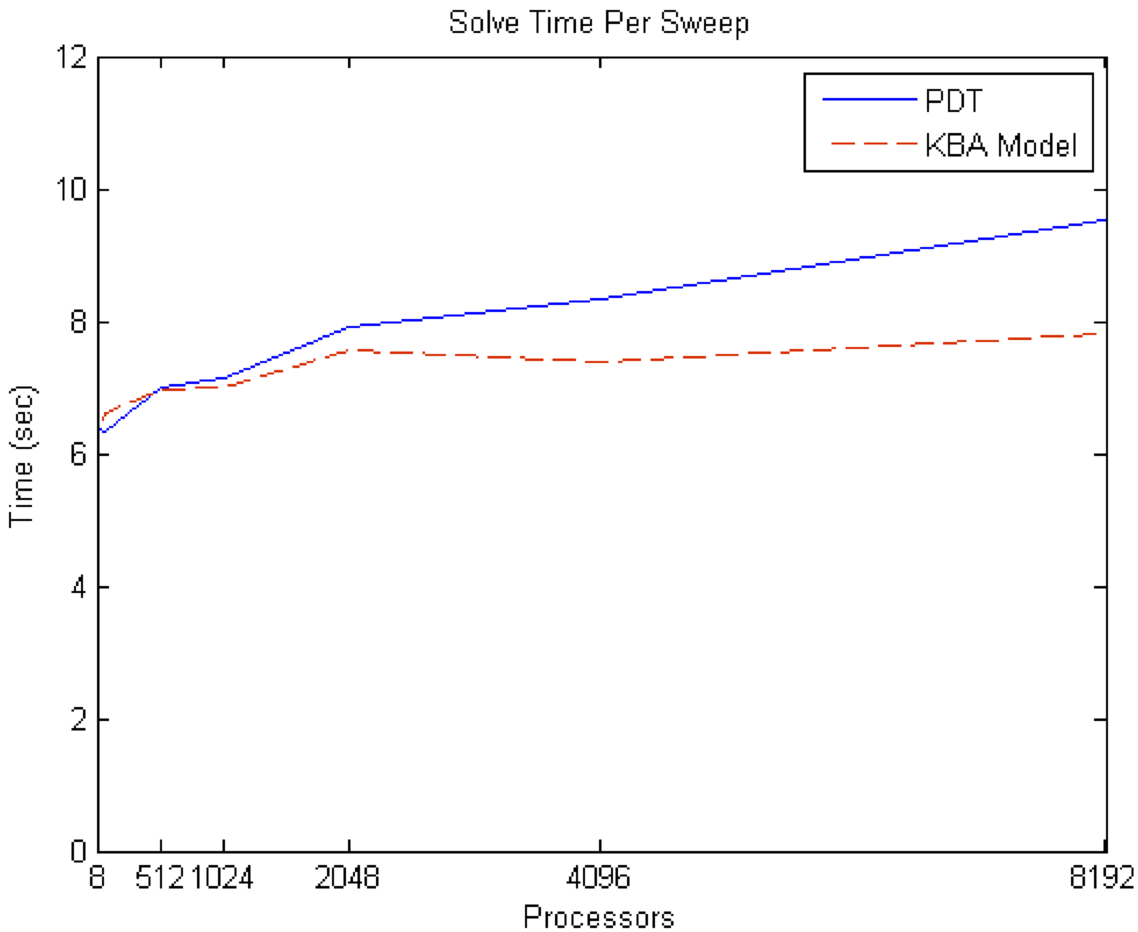




zip(prA, prB, Zipper(4,32,4));

- pRanges are sweeps in particle transport application
- Reflective materials on problem boundary create dependencies
- Composition operator will allow easy composition

Sweep Performance



- Weak scaling keeps number of unknowns per processor constant.
- Communication increases with processor count.
- KBA Model shows performance of perfectly scheduled sweep
- Divergence after 2048 processors due to non-optimal task scheduling

Conclusion

Parasol

- STAPL allows productive parallel application development
- pContainers and pAlgorithms
  - Application building blocks
  - Simplify development
  - Extensibility enables easy development of new components
- Composition of pContainers and pAlgorithms enable reuse
- RTS and FAST provide portability and adaptability