APRIL: A Run-Time Library for Tape Resident Data

Gokhan Memik, Mahmut T. Kandemir, Alok Choudhary, and Valerie E. Taylor

Center for Parallel and Distributed Computing
Northwestern University, Evanston, IL

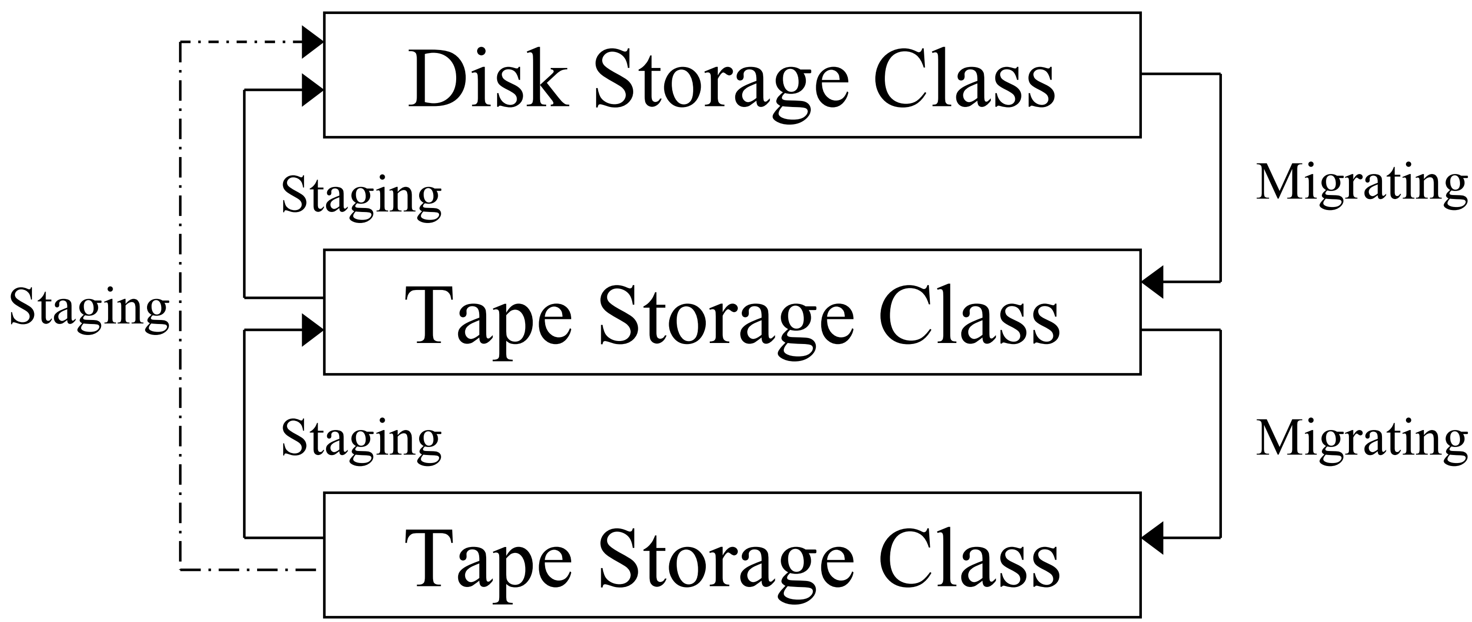
Outline

- Motivation
- Introduction
- Single Processor Data Access
- Multiple Processor Data Access
- Implementation Details
- User Interface
- Experiments
- Conclusion and Future Work

Motivation

- 3 - 30 Tbytes of simulation data for a run, 3 Petabytes of archive capacity
- Systems with several levels of storage hierarchy (e.g. HPSS)
- Tapes
  - Poor performance for random access

HPSS Overview



Introduction

- APRIL
  - Convenient user interface
  - Additional optimizations: Sub-filing, Multiple Collective I/O
- User view
  - The data is an n-dimensional matrix
  - Coordinates for a data access
  - Control over the location of data

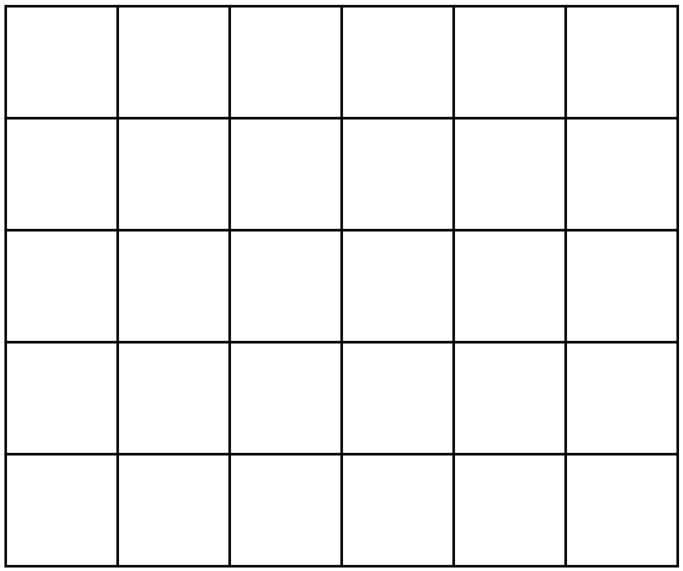
Additional Optimizations

- Sub-Filing
  - The global file is divided into *chunks*
  - User unaware of the sub-filing for ease-of-use
- Multiple Collective I/O
  - Accessing several files with a single I/O call

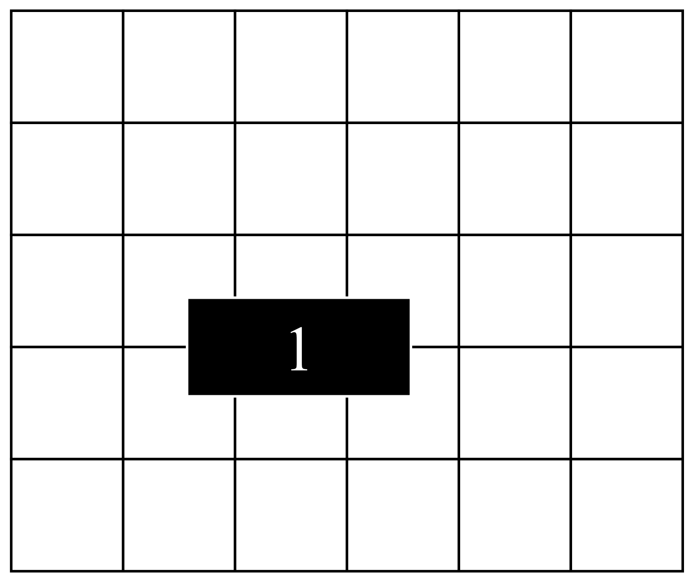
Single Processor Data Access

- Determine the chunks satisfying the access pattern (*cover*)
- Transfer the chunks (that are not already on disk) from tape to disk
- Copy the corresponding elements of the chunk to the buffer

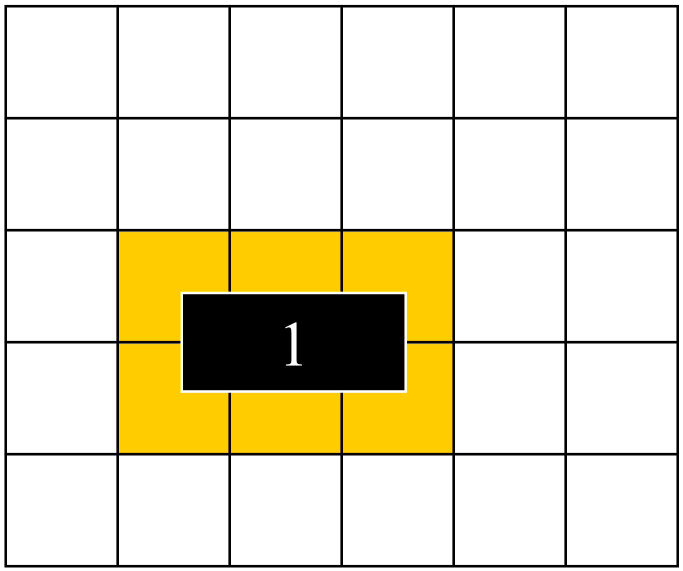
Single Processor Data Access



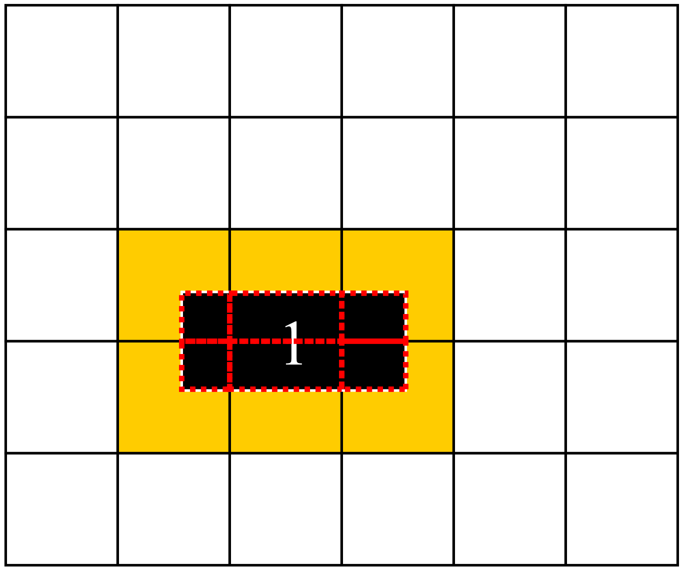
Single Processor Data Access



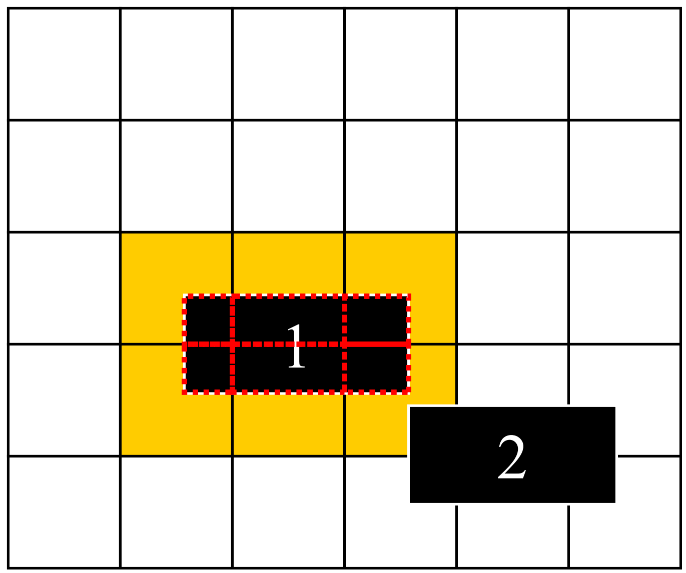
Single Processor Data Access



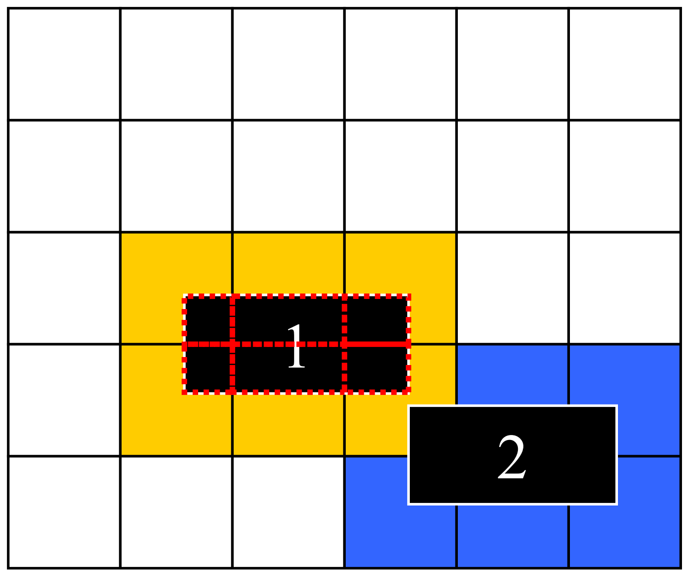
Single Processor Data Access



Single Processor Data Access



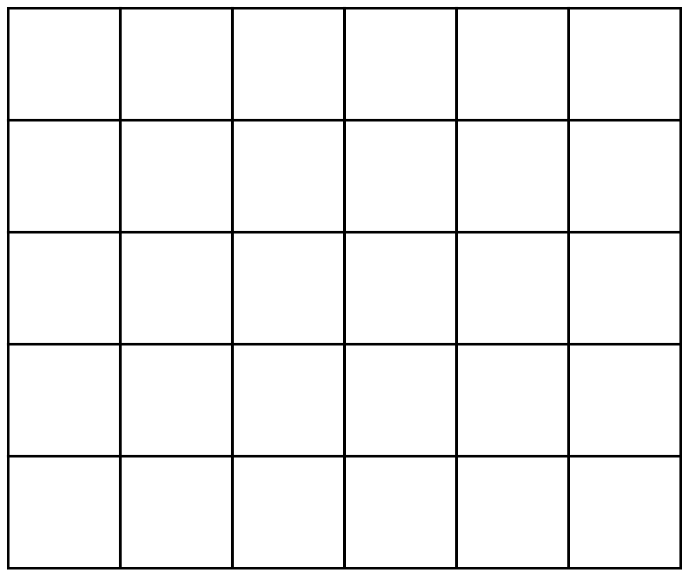
Single Processor Data Access



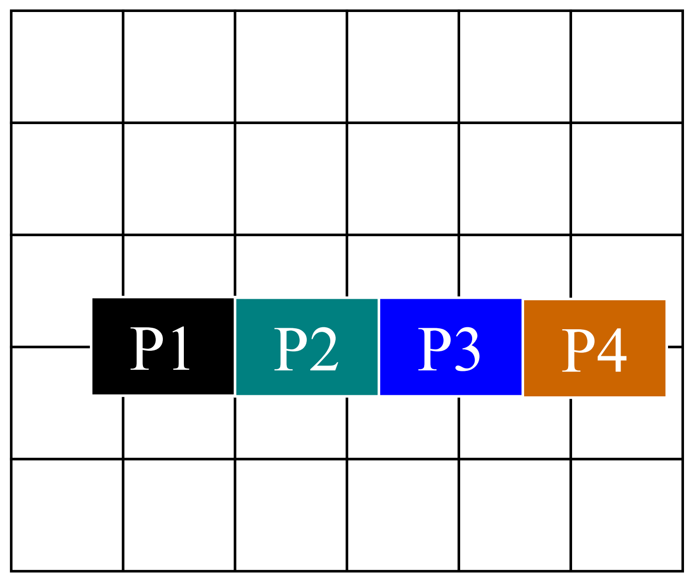
Multiple Processor Data Access

- Three-phase I/O
  - Determine collectively which chunks to be read by which processor
  - Perform I/O
  - Communicate the parts belonging to other processors
- Goal:
  - Each processor reads same number of chunks in parallel
  - Least amount of communication

Multiple Processor Data Access



Multiple Processor Data Access



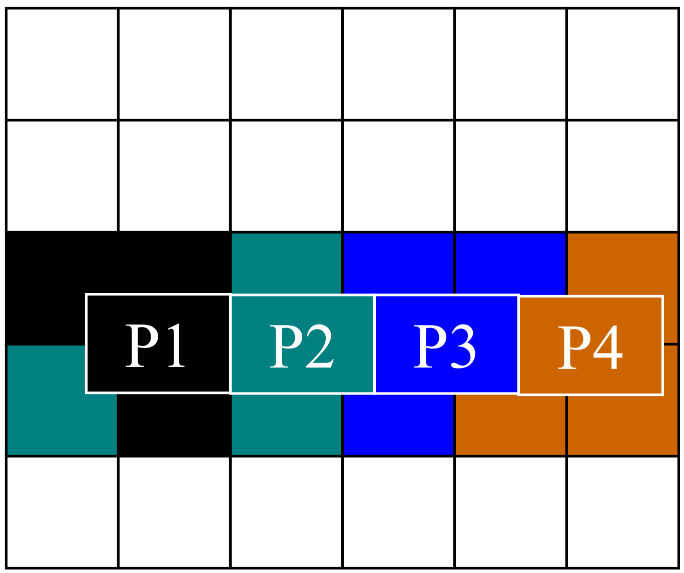
Assigning the files to processor

- Instance of Assignment Problem
  - Can use LP-relaxation
  - Too slow to solve, need heuristics
- Simplicity assumption:
  - Number of files divisible by number of processors (= k)

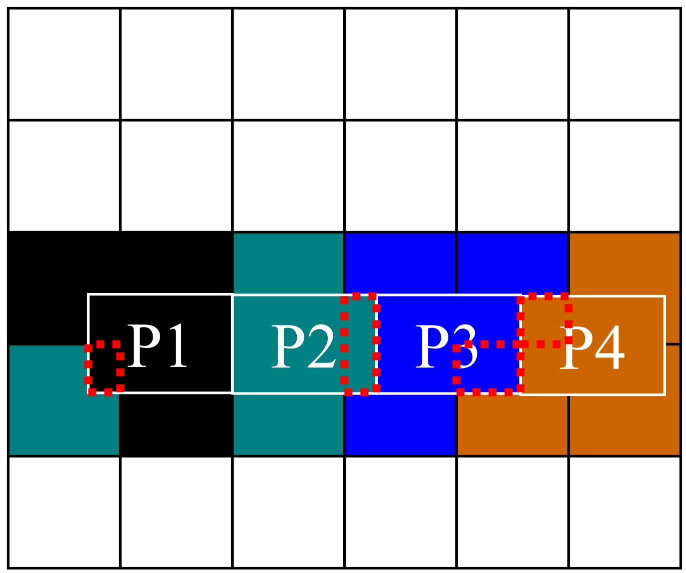
Heuristic (Greedy Algorithm)

- List1: Sort all accesses to a file according to the file size accessed (for each processor involved)
- List 2: Sort globally the access sizes
- Try to assign the largest access (from List 2) to the accessing processor
  - If processor has less than k files, assign the processor
  - Else try the next processor (on List 1) until you assign
- Update and go to the third step

Multiple Processor Data Access



Multiple Processor Data Access



Implementation

- HPSS and MPI-IO
- Postgres95 to store the information about the file and the chunks
- Database accessed only when a file is opened or closed

User Interface

- Initialization/Finalization Routines
  - T_Initialize, T_Finalize
- File Manipulation Routines
  - T_Open, T_Close, T_Remove
- Array Access Routines
  - T_Read_Section, T_Write_Section
- Stage/Migration Routines
  - T_Stage_Section, T_Prefetch_Section, T_Migrate_Section

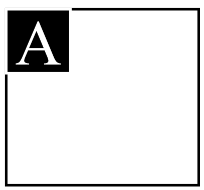
User Interface (continued)

- Data is seen as an n-dimensional matrix
- Read Example
  - T_Read_Section (T_File *fd, void *buffer, int *start_coordinate, int *end_coordinate)
    - fd: File pointer to the global file
    - buffer: Buffer where APRIL puts the result
    - Start_coordinate: Start coordinate of the section to be read for each of the dimensions of the file
    - End_coordinate: Similar to start_coordinate

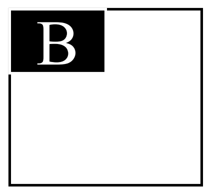
Experiments

- HPSS at the San Diego Supercomputing Center (SDSC)
- SRB to access HPSS files
- Database and the application at Northwestern University
- File size: 50000 x 50000 floating points (20 GB total data)
- Chunk Size: 2000 x 2000 floating points (32 MB)

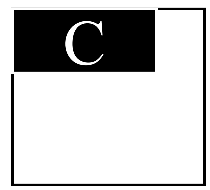
Access Patterns



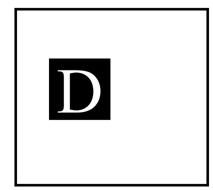
(0,0)-
(1000,1000)



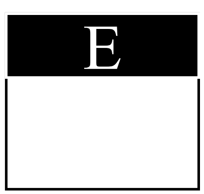
(0,0)-
(4000,1000)



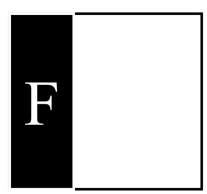
(0,0)-
(24000,1000)



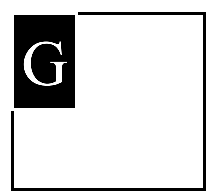
(5000,5000)-
(6000,6000)



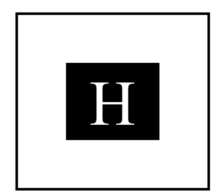
(0,0)-
(50000,80)



(0,0)-
(80,50000)

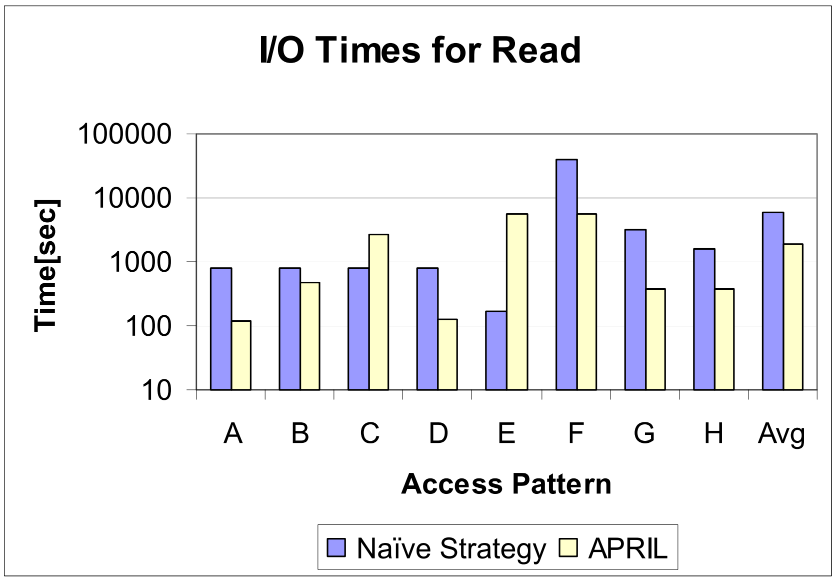


(0,0)-
(1000,4000)

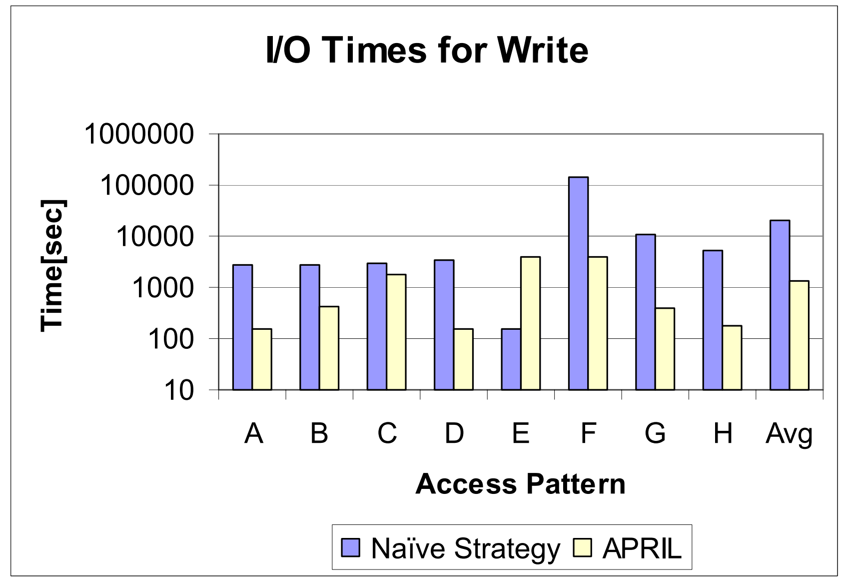


(6000,6000)-
(8000,8000)

Performance Gains



I/O Times for Read operations



I/O Times for Write operations

File Size: 50000x50000 floating points
Chunk Size: 2000x2000 floating points

Conclusion and Future Work

- Easy-to-use interface for sequential and parallel applications
- Significant performance improvements
- Automatic prefetching

http://www.ece.nwu.edu/~memik