Tuning the Performance of I/O-Intensive Parallel Applications

Anurag Acharya

Mustafa Uysal, Robert Bennett

Assaf Mendelson, Mike Beynon, Joel Saltz, Jeff Hollingsworth

Alan Sussman

Dept of Computer Science

University of Maryland

College Park MD 20742

acha@cs.umd.edu

May 27, 1996

Overview

Overall conclusions

Our applications

Our configuration

Tuning required by different applications

Evidence from our experiments

Evidence from others' experiments

Overall Conclusions

Code restructuring provides the first-order effects

- loop transformations
- locality
- request coalescing

Complex I/O interfaces less important

- loop transformations can eliminate complex patterns

Knowledge about future I/O often available

- loop splitting is a key operation

Careful scheduling needed on peer-peer systems

- conflict between computation and I/O

Our Applications

Satellite data processing:

- **pathfinder** – current in production use at NASA Goddard
  - primary program for AVHRR processing
  - representative of a large class
    - *gimms*, *SeaWIFS*, *LAS/ADAPS*, *MODIS*

Sparse Cholesky factorization:

- out-of-core
- **partitioner** and **factor**
- symmetric positive-definite

Our Configuration

16-processor IBM SP-2

Two SCSI-2 chains per processor

- three IBM Starfire 7200 disks per chain

Max application-level I/O bandwidth

- using raw disk interface: 25MB/s per node (total: 400 MB/s)
- using filesys interface: 16.8 MB/s per node (total: 270 MB/s)
- measured using multi-disk version of **iozone**

High-speed interconnect

- 40 MB/s bidirectional link per processor
- can be mapped into user-space
- max b/w measured with 128 KB messages: 32 MB/s

Disks and networks share peripherals bus

- 80 MB/s

Pathfinder

Input:

- satellite data from multiple orbits, ancillary data
- 1 GB for one day

Output:

- composite twelve-band image of the world
- total vol: 220 MB

Intermediate image:

- out-of-core, total I/O vol: 28 GB

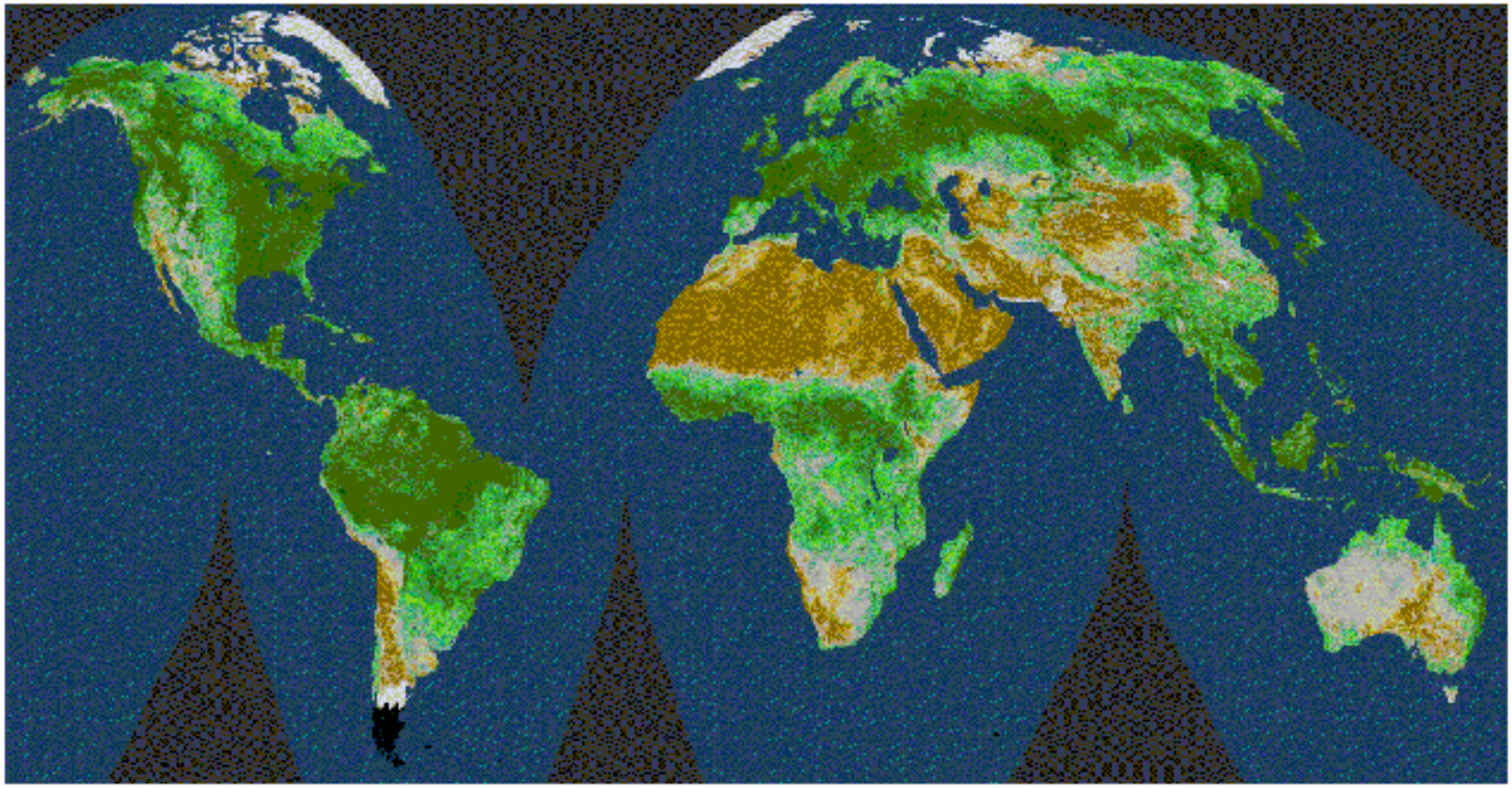
Original sequential code:

- took 18,800 sec on RS/6000
- 76% time waiting for I/O

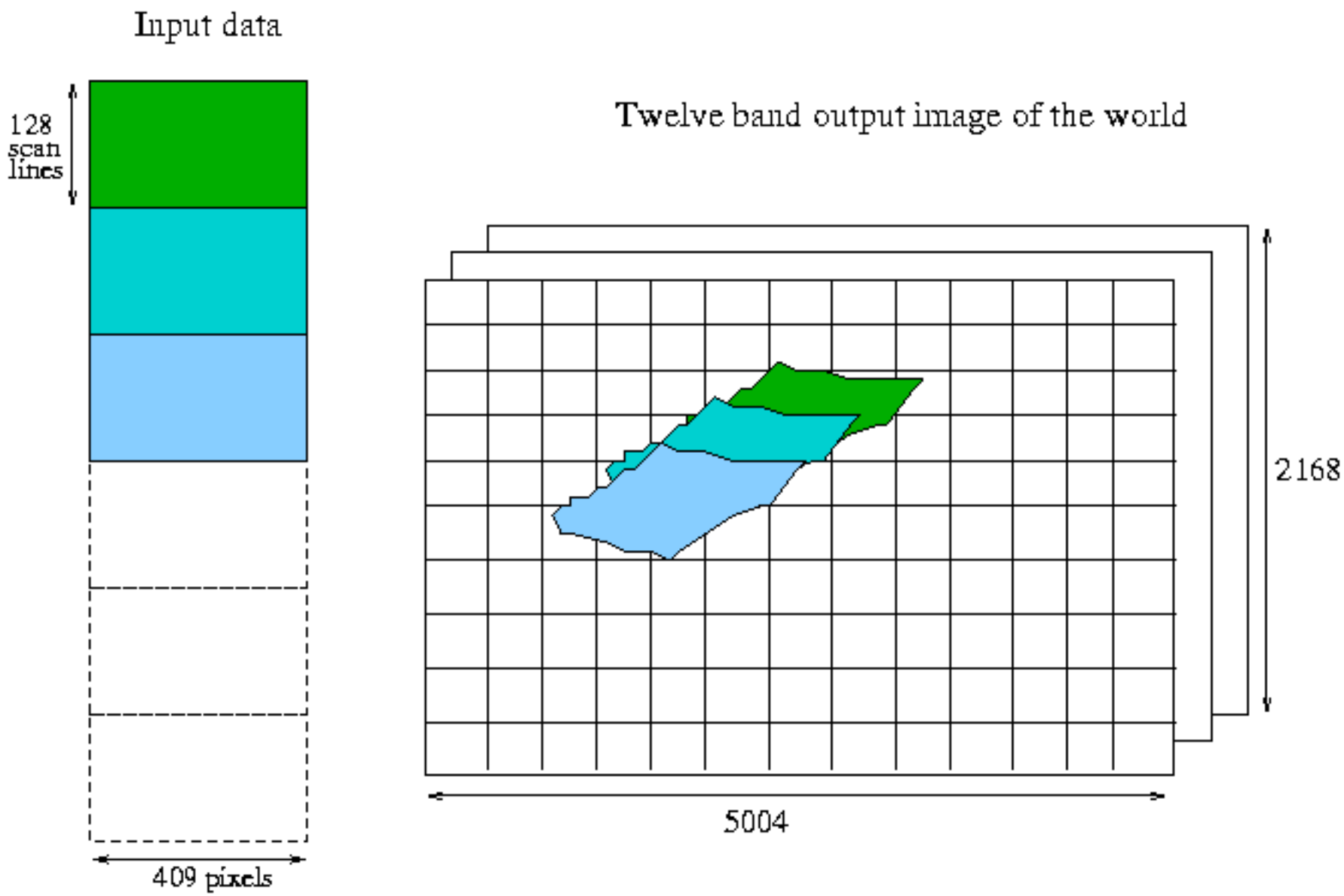
Final parallel version on 12 processors

- took 1200 sec on SP-2
- 10-15% time waiting for I/O
- max aggregate application-level I/O rate: 644 MB/s
- max end-to-end I/O bandwidth – 26 MB/s

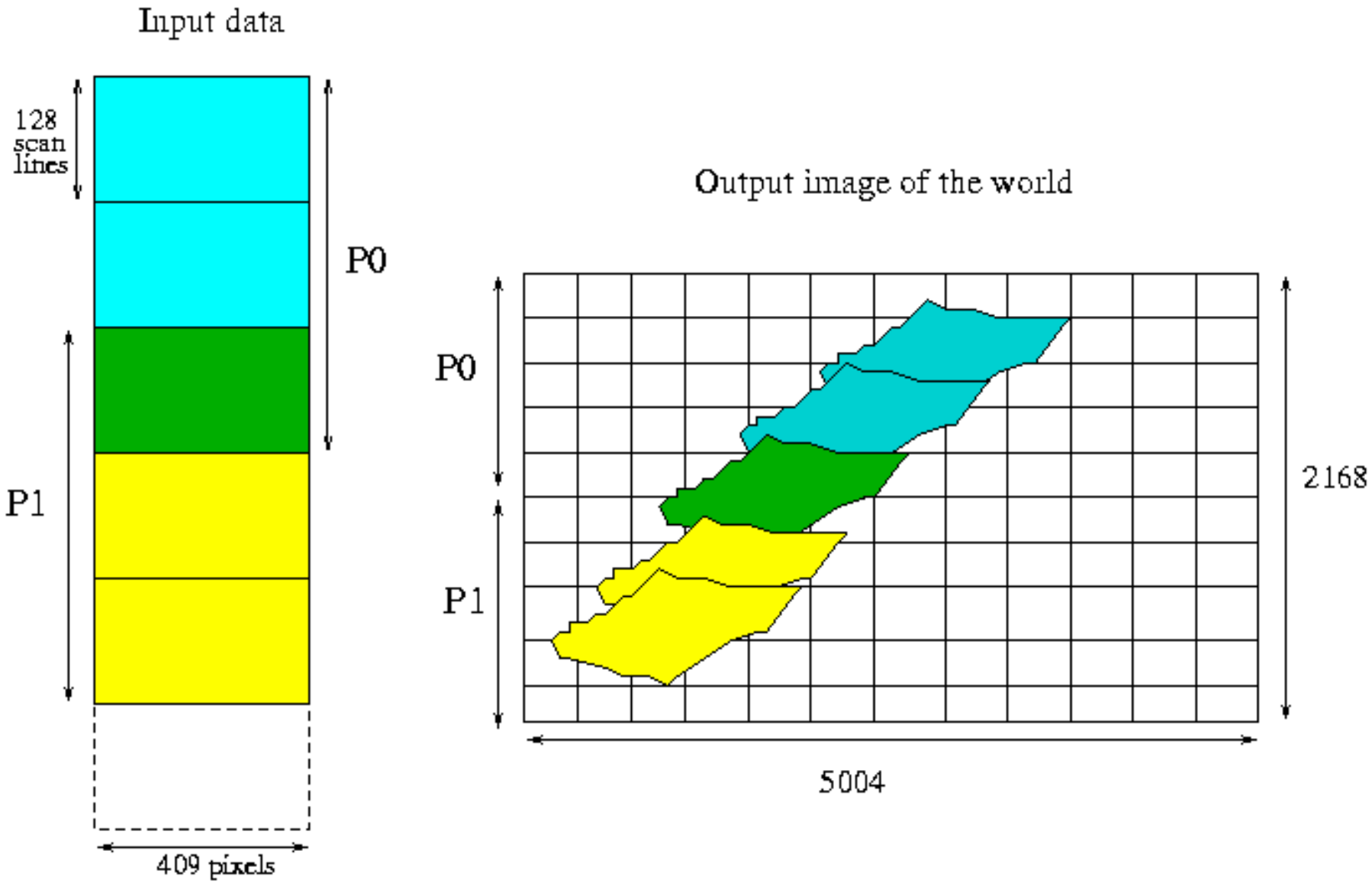
Global vegetation image from Pathfinder



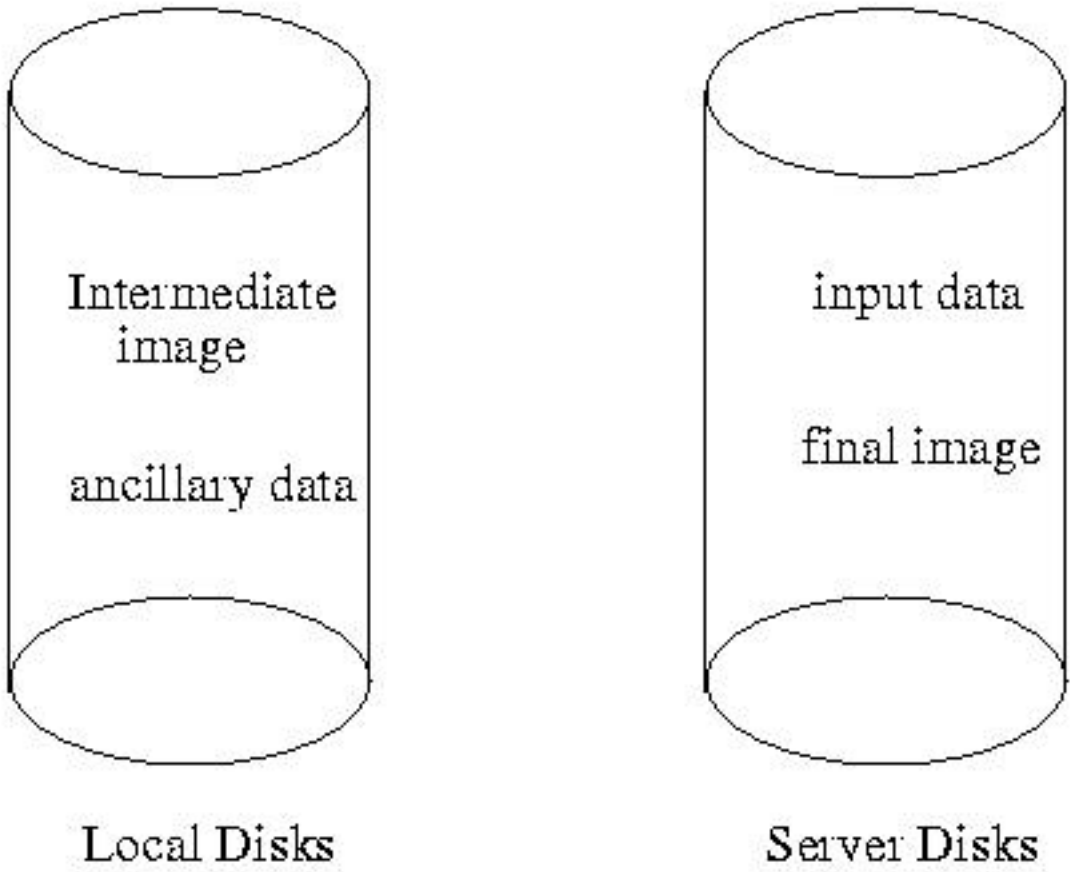
Pathfinder

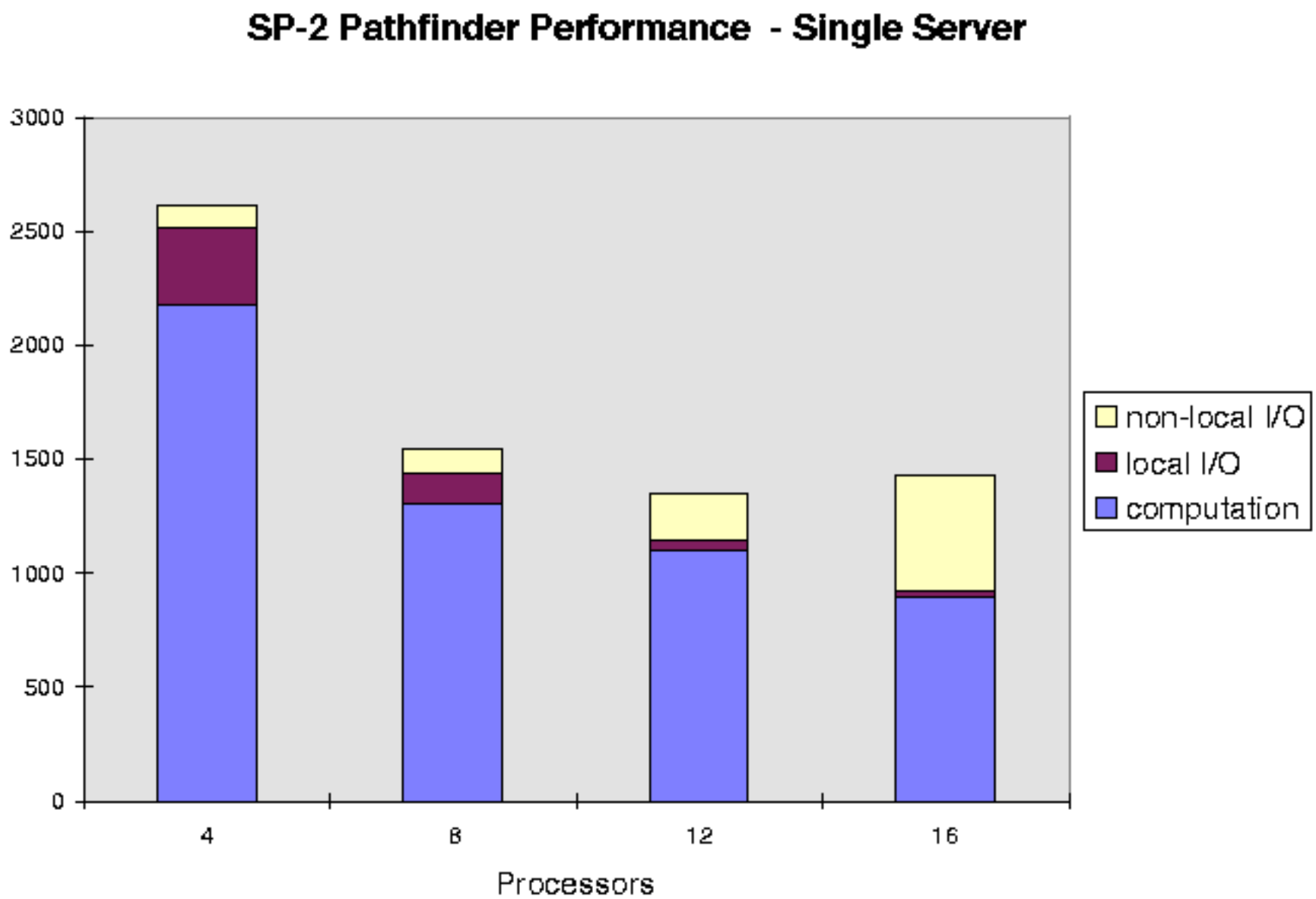

Input data
128
scan
lines
409 pixels
Twelve band output image of the world
2168
5004

Partitioning for Pathfinder



Data Placement for Pathfinder




SP-2 Pathfinder Performance - Single Server
3000
2500
2000
1500
1000
500
0
4
8
12
16
Processors
non-local I/O
local I/O
computation

SP-2 Pathfinder

Aggregate I/O Rate						
	One Server			Two Servers		
Processors	**8**	**12**	**16**	**8**	**12**	**16**
Input	25	26	15	46	60	66
Output	34	42	63	56	108	126
Intermed read	246	510	634	193	453	644
Intermed write	169	118	407	147	256	372
Overall	96	104	44	122	163	161

Total I/O volume divided by time spent in I/O routines (MB/sec)

Code restructuring needed for Pathfinder

Read input in large blocks

- used to read 3.5 KB (one scan line) at a time
- now reads 512 KB at a time

Loop splitting

- needed for out-of-core read-modify-write
- pre-compute bounding box

Loop reordering

- hoist I/O to outer loop
- coalesce small I/O requests

Place out-of-core files on local disks

- no conflicts for file-cache
- local disk bandwidth 7 MB/s
- non-local disk bandwidth $\leq$ 5.7 MB/s

Place input and output files on I/O nodes

- small fraction of I/O volume ($\leq$ 10%)

Loop Tranformations for Pathfinder

```
foreach new_pixel in chunk
  read one band of old_pixel
  if (new_pixel better than old_pixel)
    foreach band in output_image
      read value for old_pixel
      update with value from new_pixel
      write the value back to disk
    endfor
  endif
endfor
```

```
read bounding box of old_pixels
foreach pixel in chunk
 if (new_pixel better than old_pixel)
  mark bitmap
 endif
endfor

compute bounding box for read-modify-write

foreach band in output_image
  read bounding box for band
  foreach new_pixel in chunk
    if (bitmap value set)
      update with value from new_pixel
    endif
  endfor
  write bounding box back to disk
endfor
```

Knowledge about future I/O (Pathfinder)

Input data partitioned at start-up time

- processors subsample input data to build partition
- essentially loop splitting
- complete future knowledge about input available
- partition is irregular

Intermediate I/O is data-dependent

- bounding box for updates computed by loop splitting
- same bounding box needed for twelve read-modify-writes

Knowledge about future I/O (Pathfinder)

Input data partitioned at start-up time

- processors subsample input data to build partition
- essentially loop splitting
- complete future knowledge about input available
- partition is irregular

Intermediate I/O is data-dependent

- bounding box for updates computed by loop splitting
- same bounding box needed for twelve read-modify-writes

Sparse Cholesky Factorization

partitioner computes *fill-in* & distributes data

partitioner input :

takes output from symbolic factorization as input

non-zeroes and structure of original matrix

sparsity structure of the factor

factor:

order of operations known (elimination-tree)

pre-fetching of local and non-local data

caching of data to be re-used

Matrix	N	$\|A\|$	Partitioner I/O		Factorization I/O	
			Read	Write	Read	Write
skirt	45361	1.3×10^6	381	403	20,200	377
sara-1	44856	2.6×10^6	488	534	49,000	509
sara-2	80651	2.9×10^6	301	1,939	220,800	838

Application I/O volumes are in MB

SP-2 Matrix Factor

- Aggregate I/O rates
- all data in "effective" MB/s

	4 nodes		8 nodes		16 nodes	
Matrix	Read	Write	Read	Write	Read	Write
skirt	47.4	83.6	87.6	172.1	121.6	242.7
sara-1	76.3	101.5	96.2	199.7	157.3	277.3
sara-2	-	-	-	-	96.1	182.7

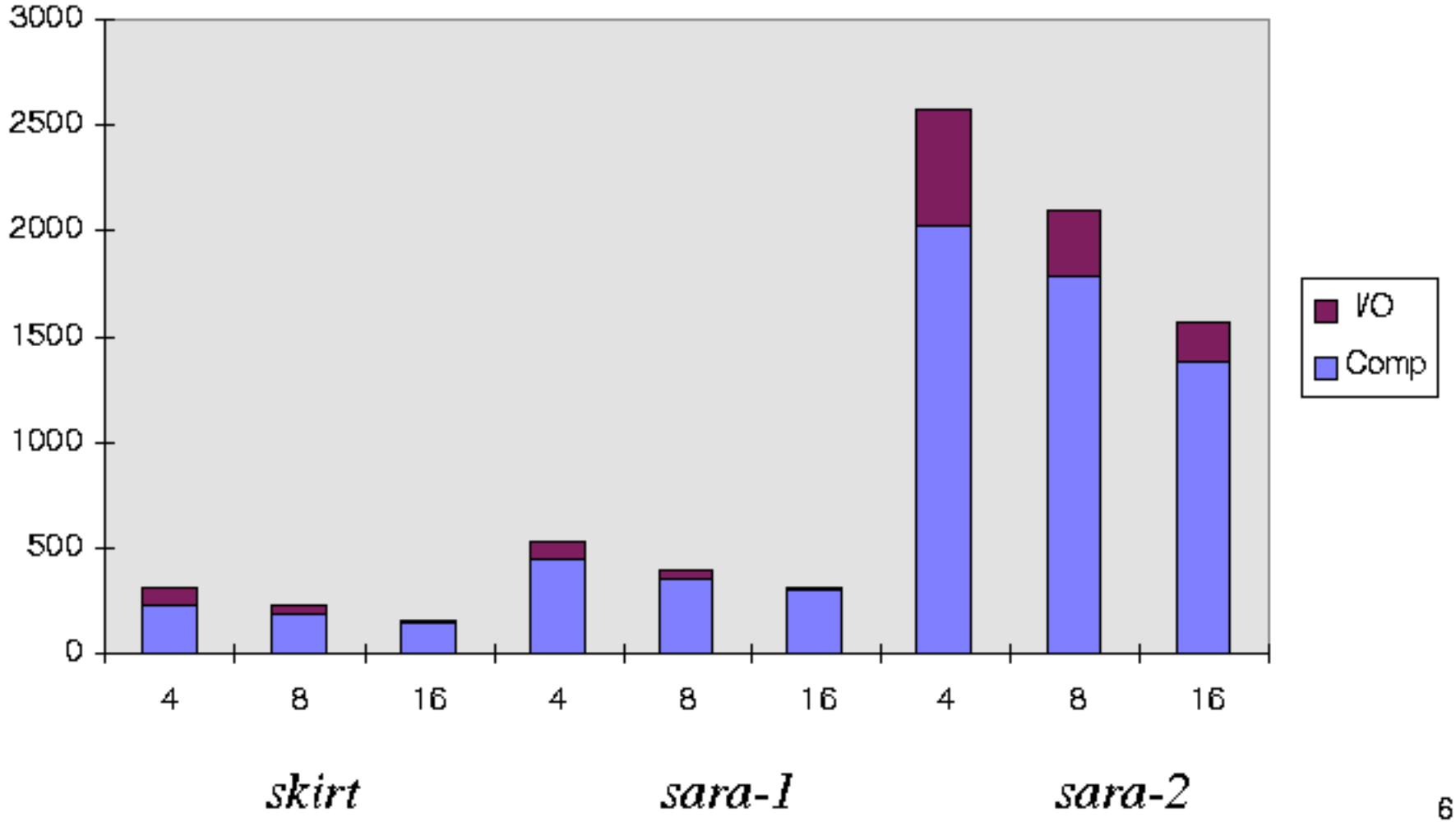
SP-2 Partitioner

- Aggregate I/O rates for partitioner
- all data in "effective" MB/s

	4 processors			8 processors			16 processors		
data	Rd-1	Rd-2	Wr	Rd-1	Rd-2	Wr	Rd-1	Rd-2	Wr
skirt	**14.1**	**19.4**	**9.9**	**43.0**	**45.8**	**19.7**	**108**	**380**	**41**
sara-1	**12.3**	**15.9**	**9.3**	**18.6**	**29.5**	**18.7**	**106**	**430**	**15**
sara-2	**20.4**	**49.6**	**2.5**	**42.1**	**76.4**	**16.4**	**103**	**297**	**15**

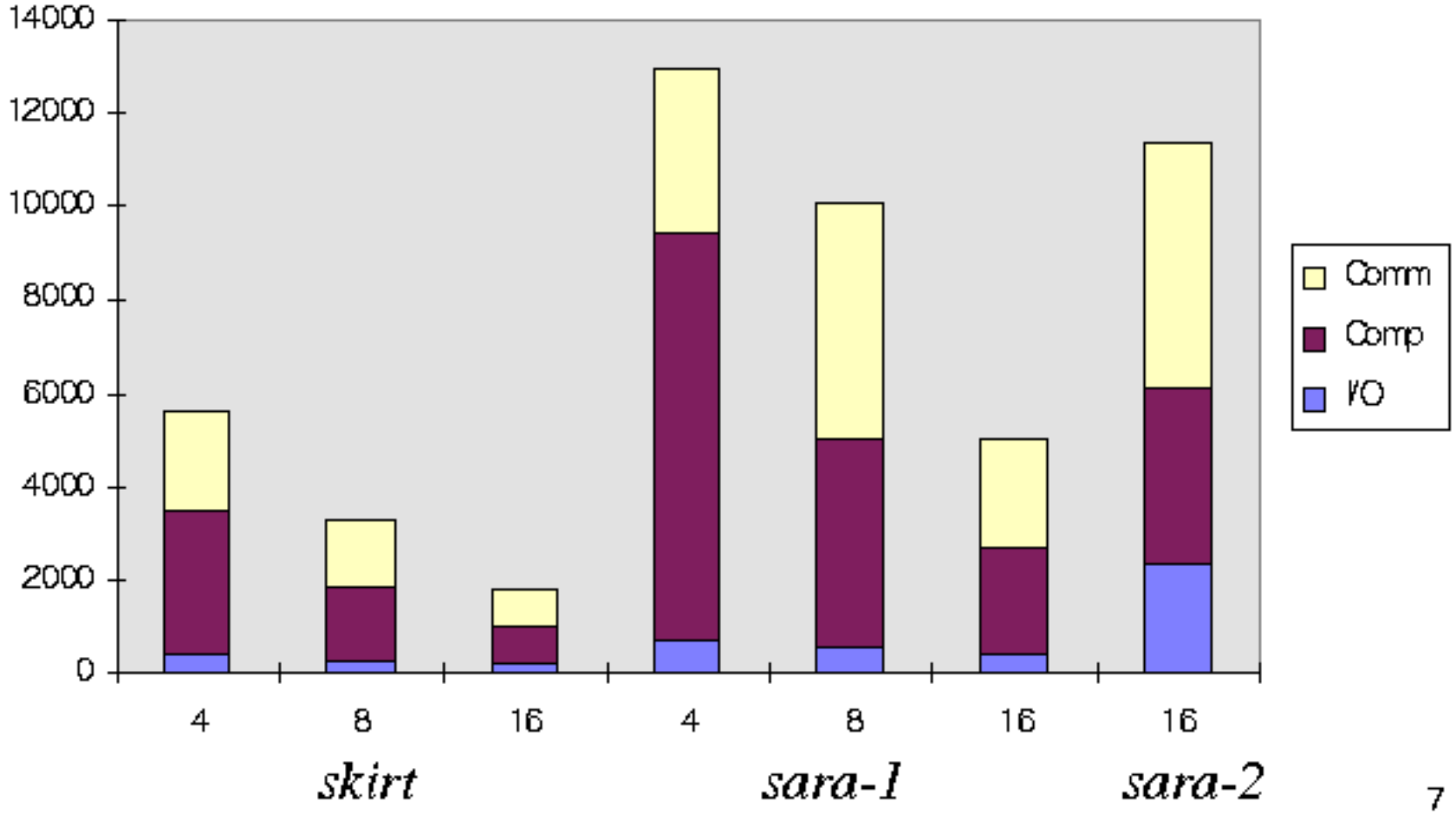
Execution Time for Partitioner

- All times are in seconds



Execution Time for Factor

- All Times are in seconds



Code structuring for Factor

Peer-peer configuration

- data distributed over all processors

Balance I/O, communication and computation

- async I/O and communication
- computation and I/O are co-routines

Server-push model of I/O

- no requests, wait for non-local data
- elimination-tree used to prefetch local and non-local data
- subject to memory and time-to-use constraints

Code Skeletons for Factor

Using future knowledge

```
loop over supernodes
  check for I/O completion

  loop over dependencies
     Issue async I/O requests for data needed
            by this and other procs
    check for data recieved from other procs
    async send any available data needed by
           other procs
    do computation for the supernode
    write updated blocks to local disk
  endloop

endloop
```

Not using future knowledge

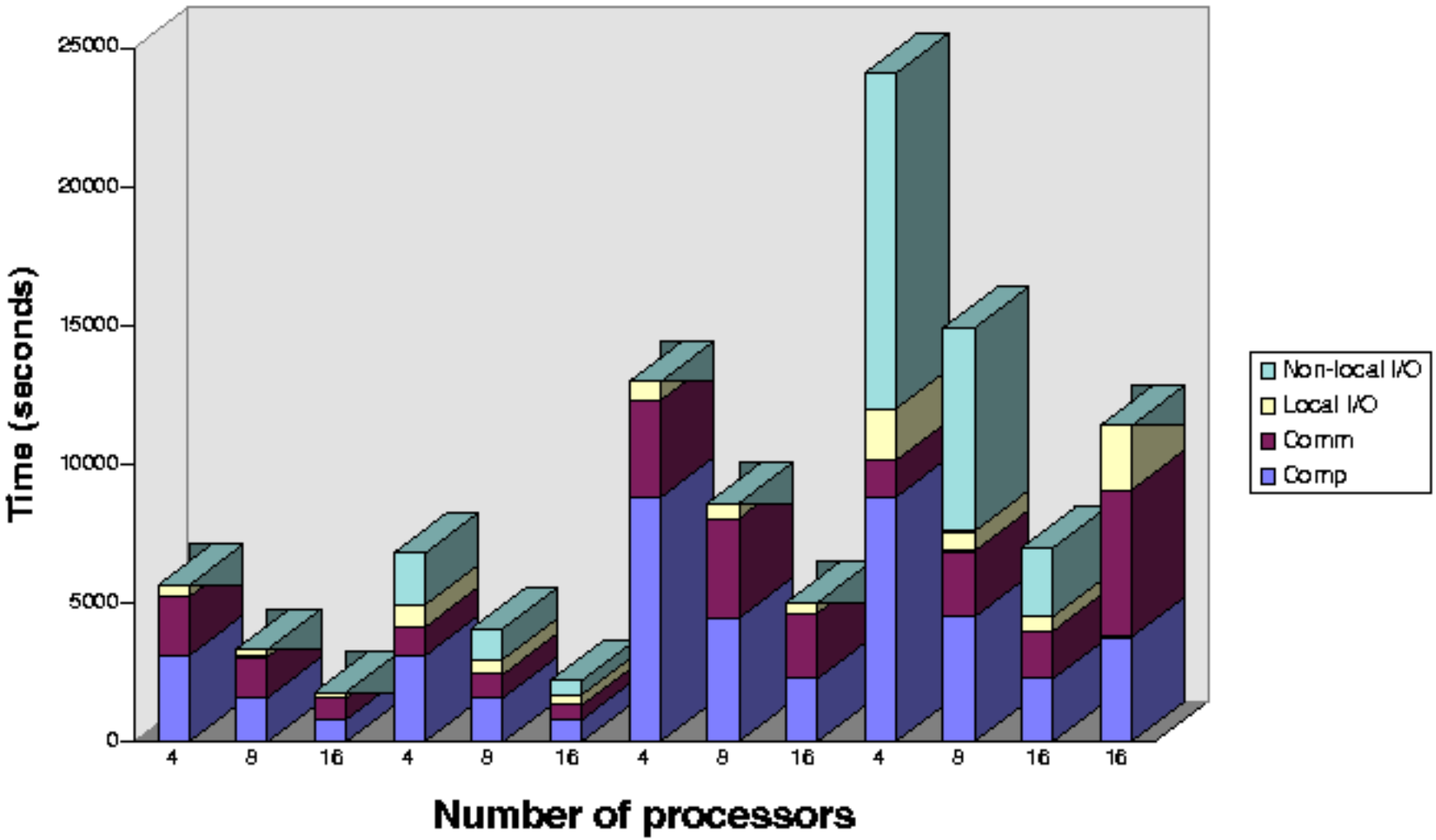
```
loop over supernodes

    loop over dependencies
      read local data and/or send request for
          remote data
      while (remote data not arrived)
         service off-proc requests
      endwhile

      do computation for the supernode
      write updated blocks to local disk
   endloop

endloop
```

Impact of future knowledge



Code restructuring in literature

Crandall et al, SC'95:

- RENDER – uses large asynchronous reads for input
  - each processor reads back what it wrote
- Hartree-Fock – needs locally stored pre-computed integrals

NAS Newsletter vol 2, Num 11 (Charbel Farhat):

- direct dense solver 100,000 unknowns
- total I/O 1.54 TB
- I/O waiting time: 65 mins (total time: 21.5 hours)
- data partitioned for locality
- new partial pivoting strategy

Toledo & Gustafson, IOPADS'96:

- 215 MFLOPS/s, I/O $\leq$ 16% of execution time
- pipelining policies to overlap I/O and computation
- new algorithms

Knowledge about future I/O

Similar experiences reported by Patterson et al [SOSP95]

- uniprocessor applications
- Hartree-Fock, XDataSlice, Sphinx, agrep, Postgres, gnuld
- loop splitting was the key

RENDER: *Crandall et al, SC'95*

Out-of-core FFT: *Sweet & Wilson U of Colorado at Denver*

Parallel PIC Simulation of Plasma: *Williams & Dimits*

- National Energy Research Supercomputer Center

Scheduling on peer-peer systems

Need control over scheduling I/O

- priority to computation: I/O backup
- priority to I/O: computation and message backup
- backup propagates

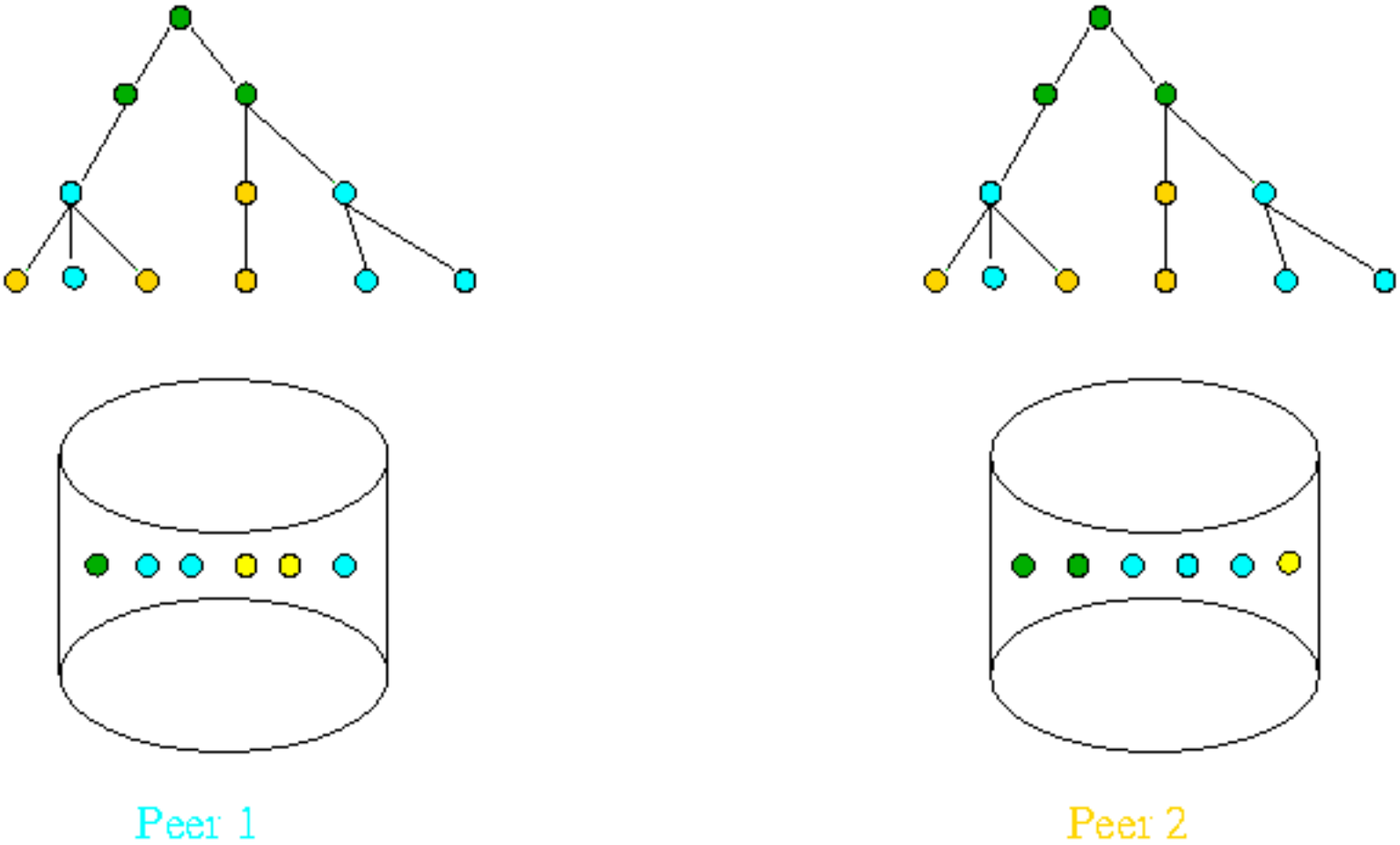
pathfinder used pre-emptive threads with polling

- occasional message takes forever
- did not achieve good performance on peer-peer

factor used co-routines with polling and server-push

- size of supernodes varies, occasional is very long
- results in long wait for nonlocal data
- idle time propagates

Information about future I/O helps

- **factor** was able to achieve decent performance
- server-push needs pacing to avoid flooding clients

Server-push I/O