MPJava: High-Performance Message Passing in Java using Java.nio

Bill Pugh
Jaime Spacco

University of Maryland, College Park

Message Passing Interface (MPI)

- MPI standardized how we pass data on a cluster
- MPI:
  - Single Processor Multiple Data (SPMD)
  - Provides point-to-point as well as collective communications
  - Is a set of library routines
  - Is an interface with several free and commercial implementations available
  - source code is portable
  - Has C, Fortran and C++ bindings, but not Java

Previous Java + MPI work:

- mpiJava (Carpenter)
  - Native wrappers to C libraries
  - Worse performance than native MPI
- jmpi
  - Pure-Java implementation of proposed standard for Java/MPI bindings
  - Also bad performance compared to native MPI

MPJava

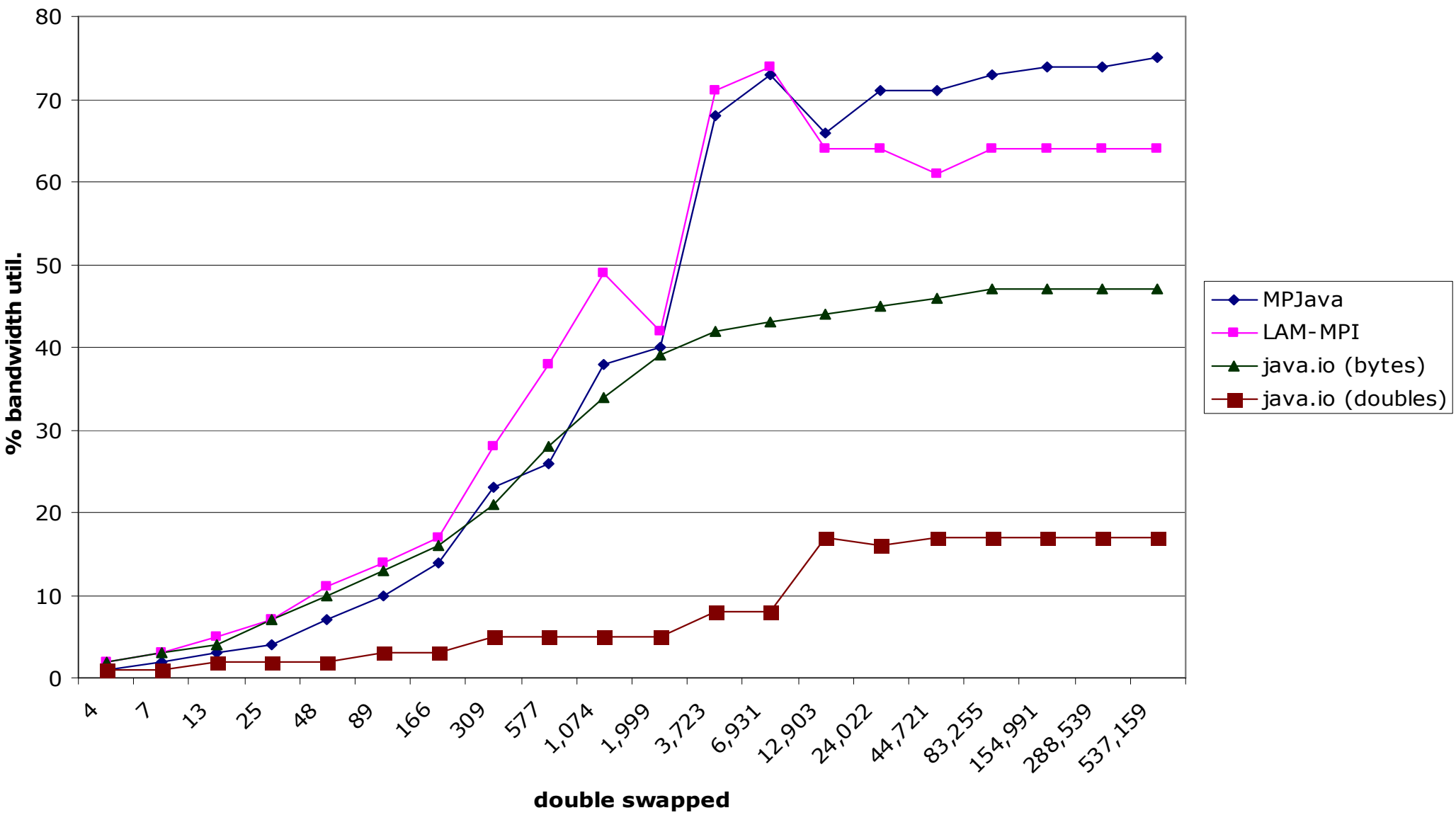
- Pure-Java Message Passing framework
- Makes extensive use of java.nio
  - *select*() mechanism
  - direct buffers
  - efficient conversions between primitive types
- Provides point-to-point and collective communications similar to MPI
- We experiment with different broadcast algorithms
- We try to use threads
  - More work needed to get this right
- Performance is pretty good

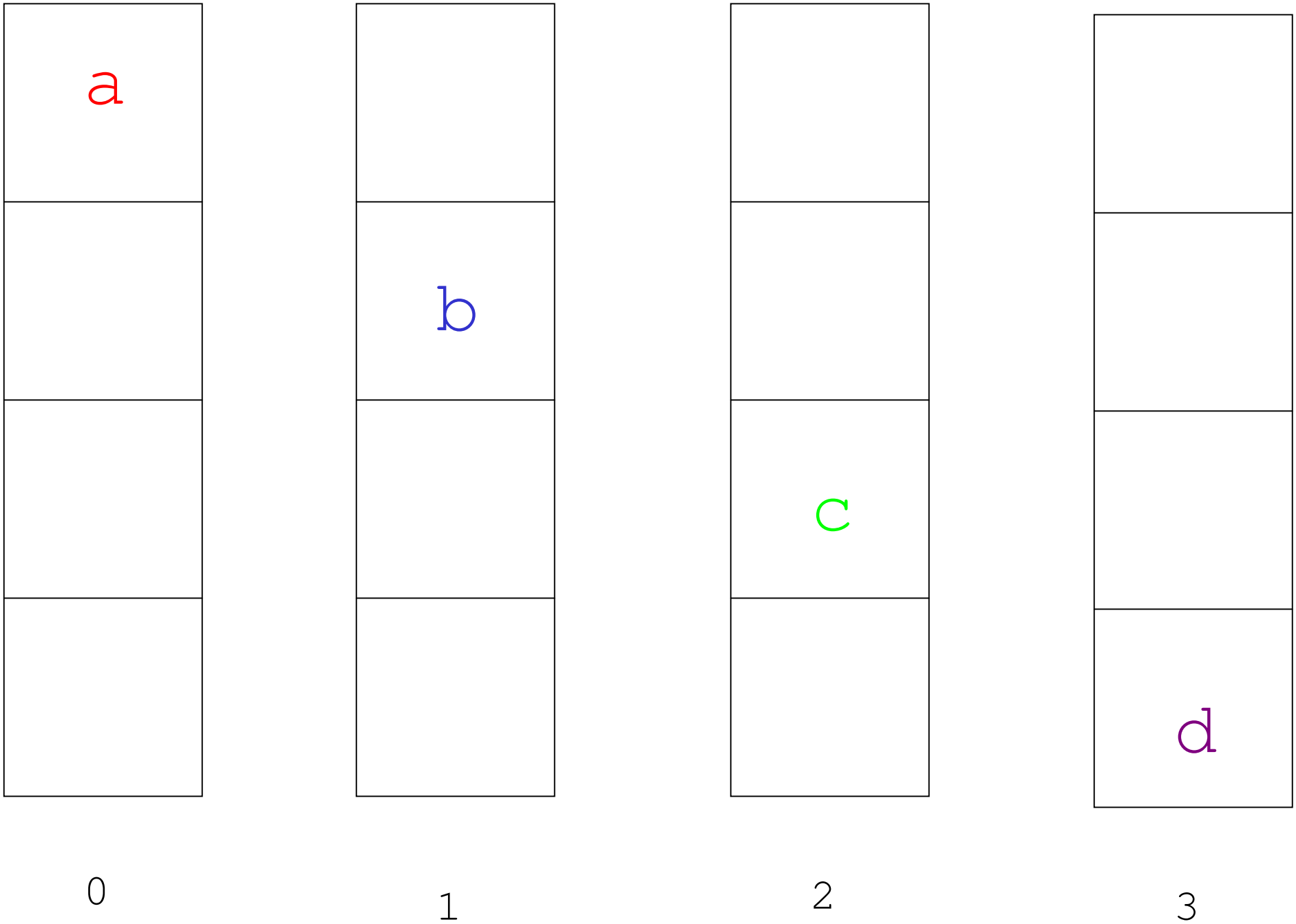
Benchmarks

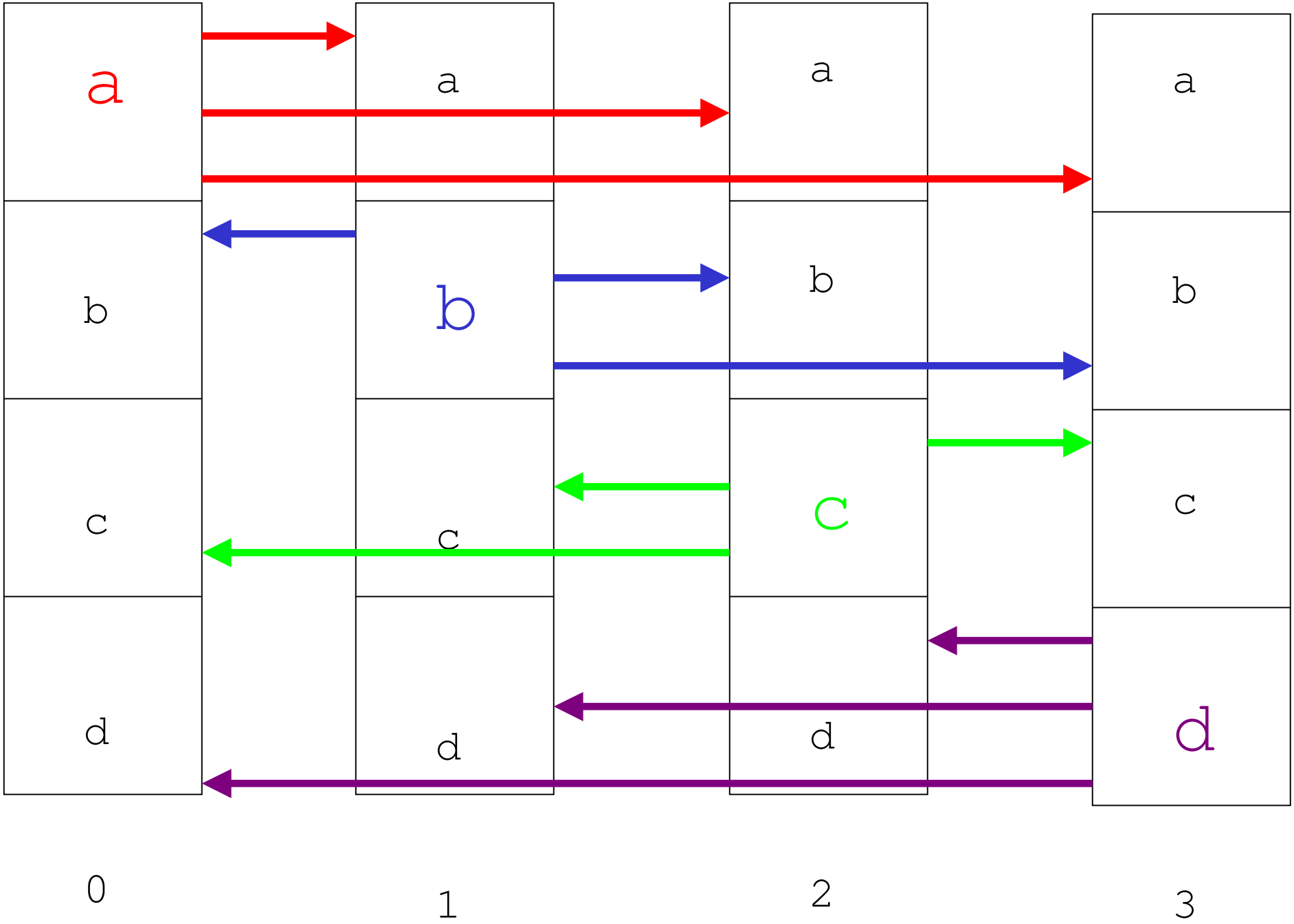
- PingPong
- Alltoall
- NAS Parallel Benchmarks Conjugate Gradient

Results

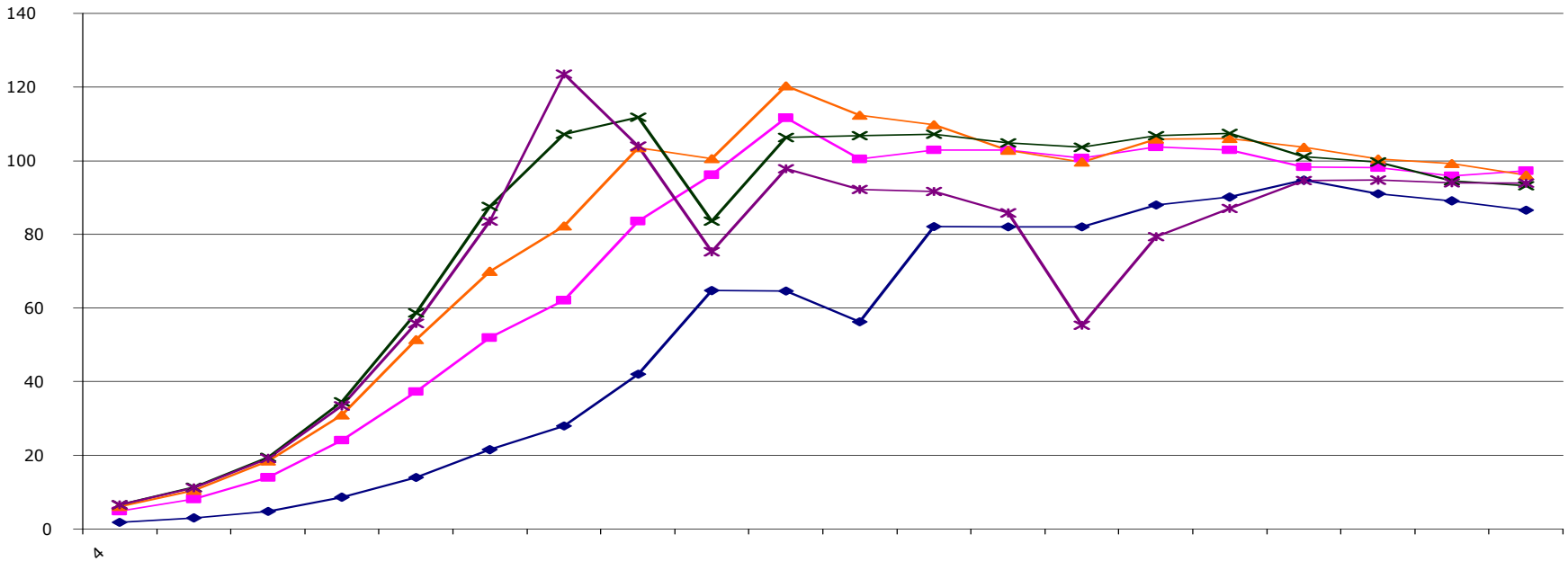
- 50 650 MHz PIII machines
- 768 MB memory
- RedHat 7.3
- two 100 Mbps channel-bonded NICs
- Fortran compiler: g77 v2.96
  - tried a commercial compiler (pgf90) but no difference for these benchmarks
- LAM-MPI 6.5.8
- JDK 1.4.2-b04


PingPong
% bandwidth util.
80
70
60
50
40
30
20
10
0
4
7
13
25
48
89
166
309
577
1,074
1,999
3,723
6,931
12,903
24,022
44,721
83,255
154,991
288,539
537,159
double swapped
MPJava
LAM-MPI
java.io (bytes)
java.io (doubles)

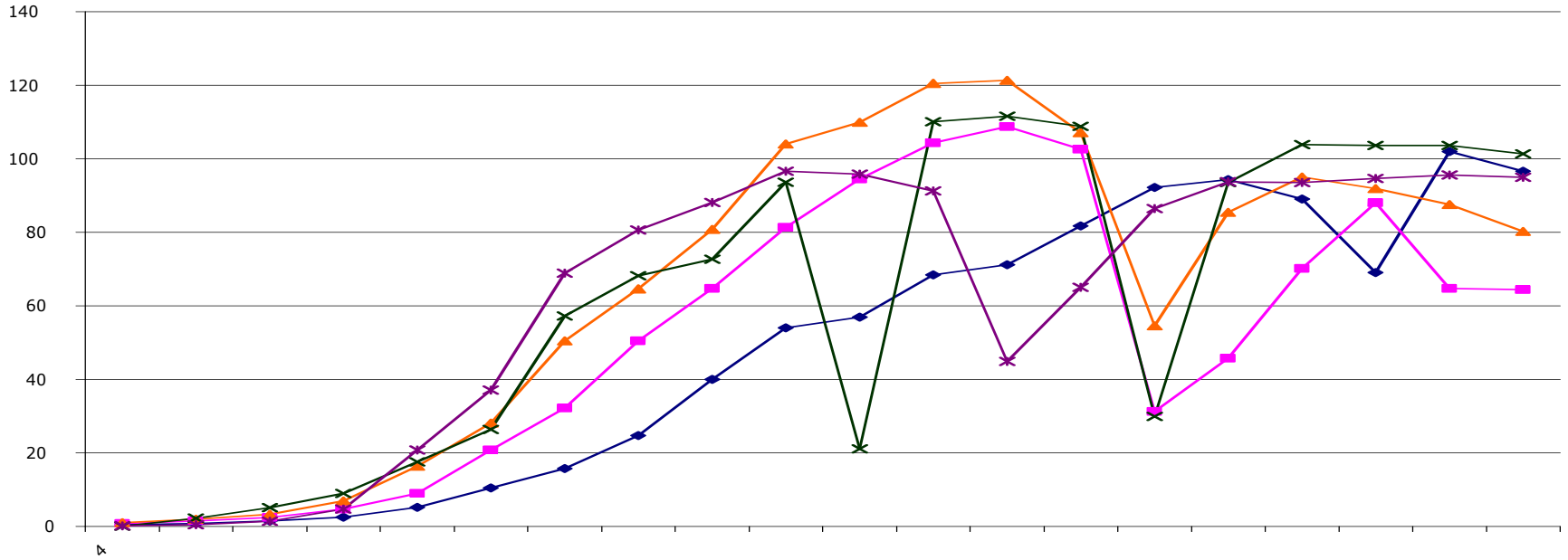

a
b
c
d
0
1
2
3

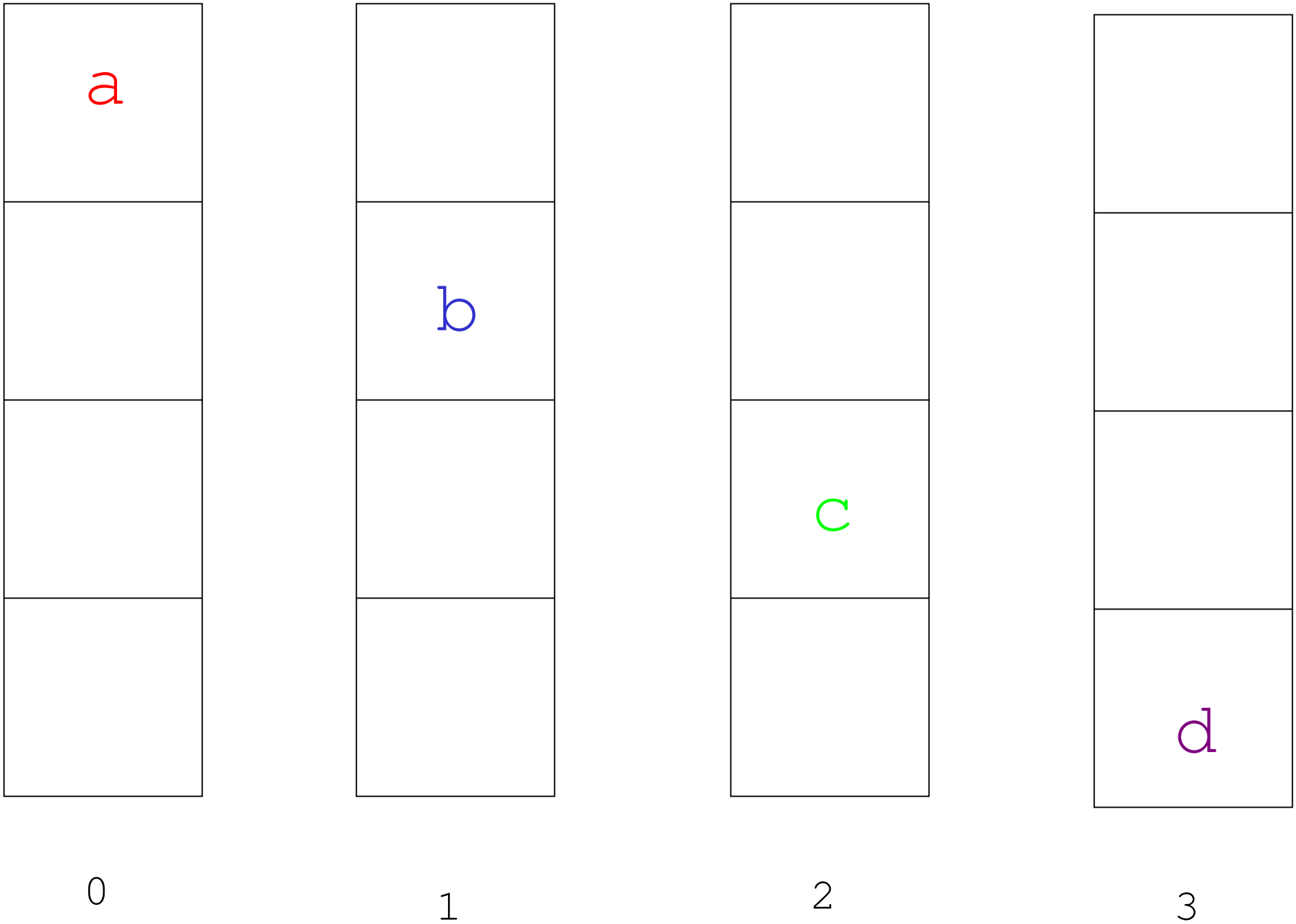

a
a
a
a
b
b
b
b
c
c
c
c
d
d
d
d
0
1
2
3

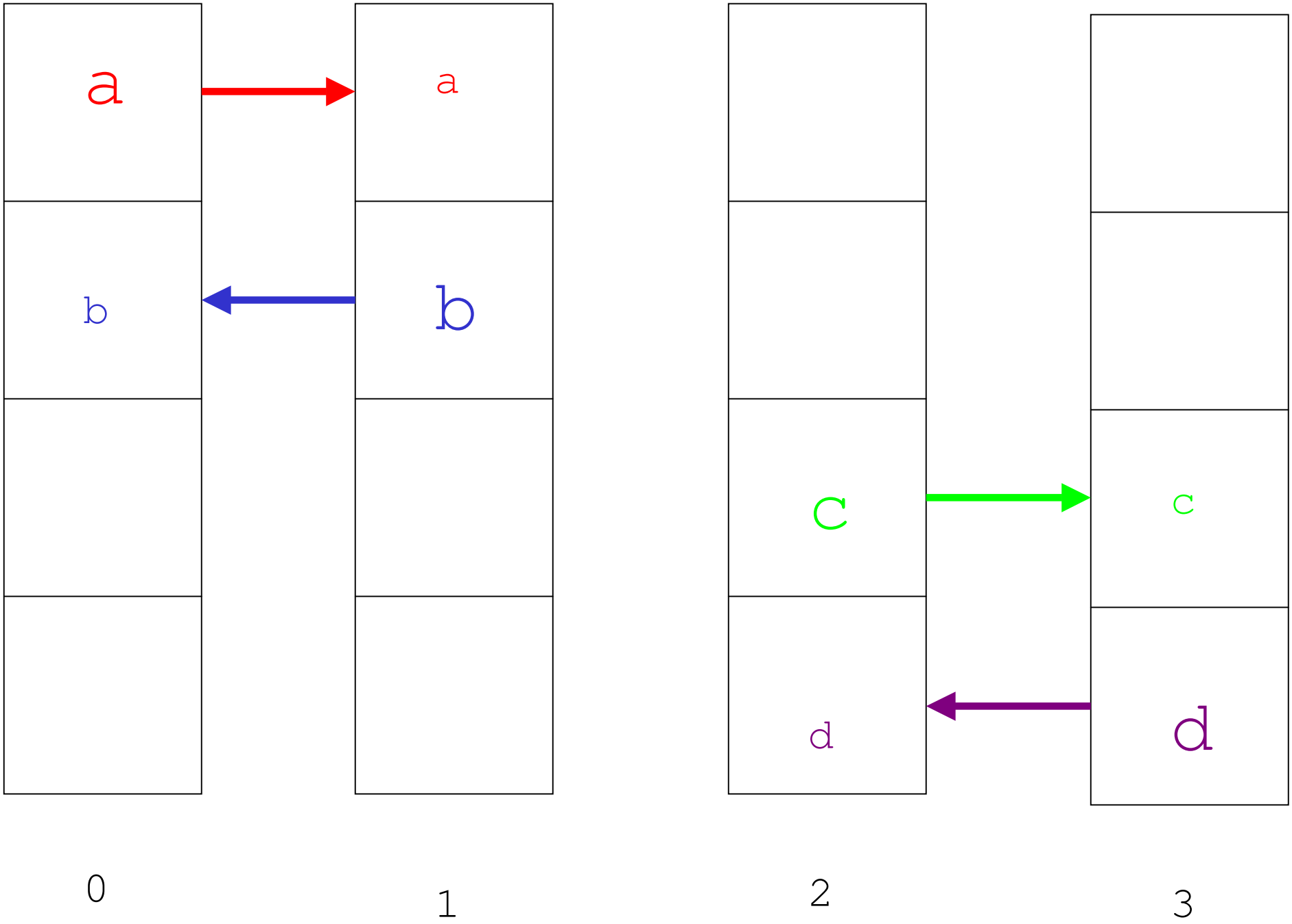
Alltoall LAM

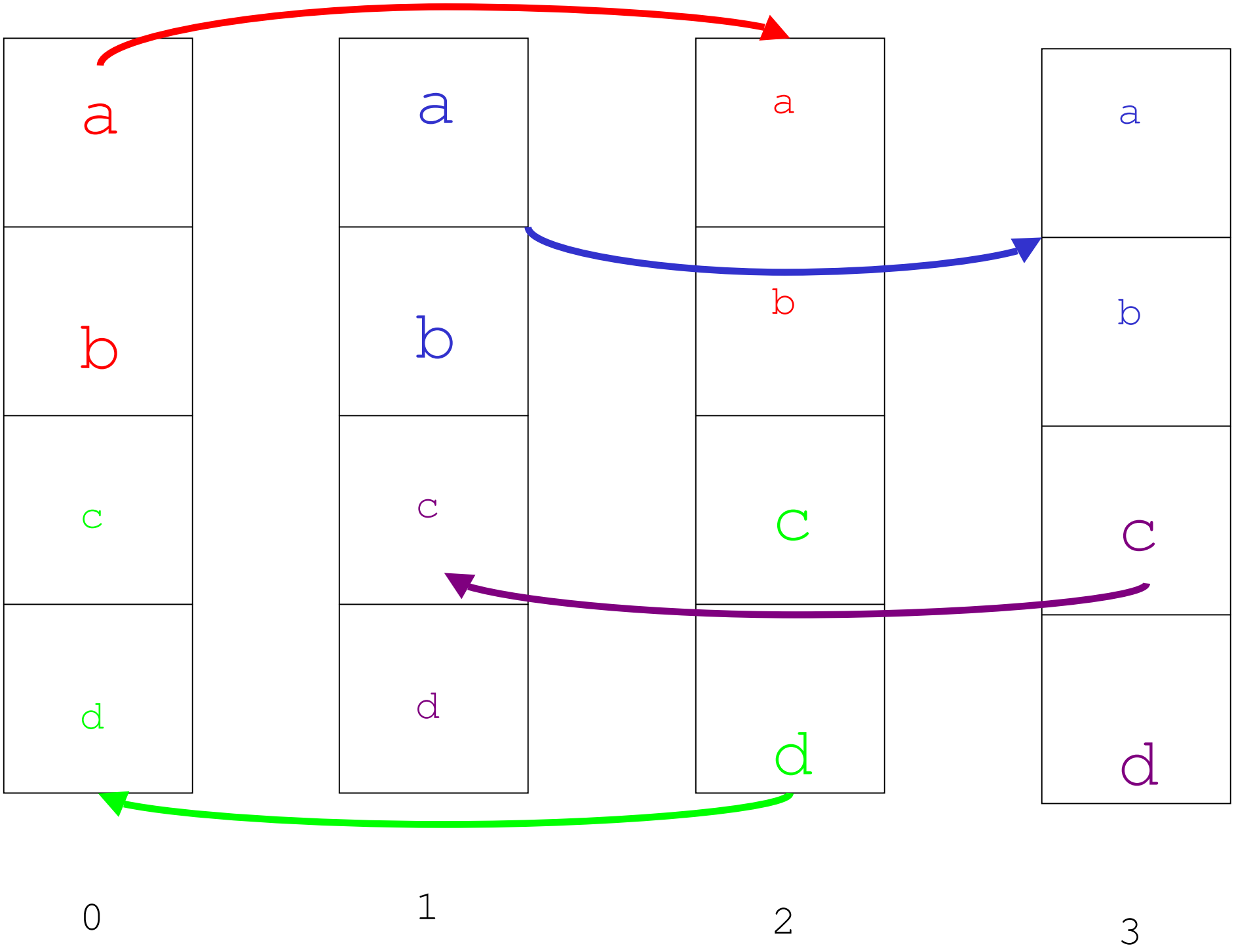


Alltoall MPJava

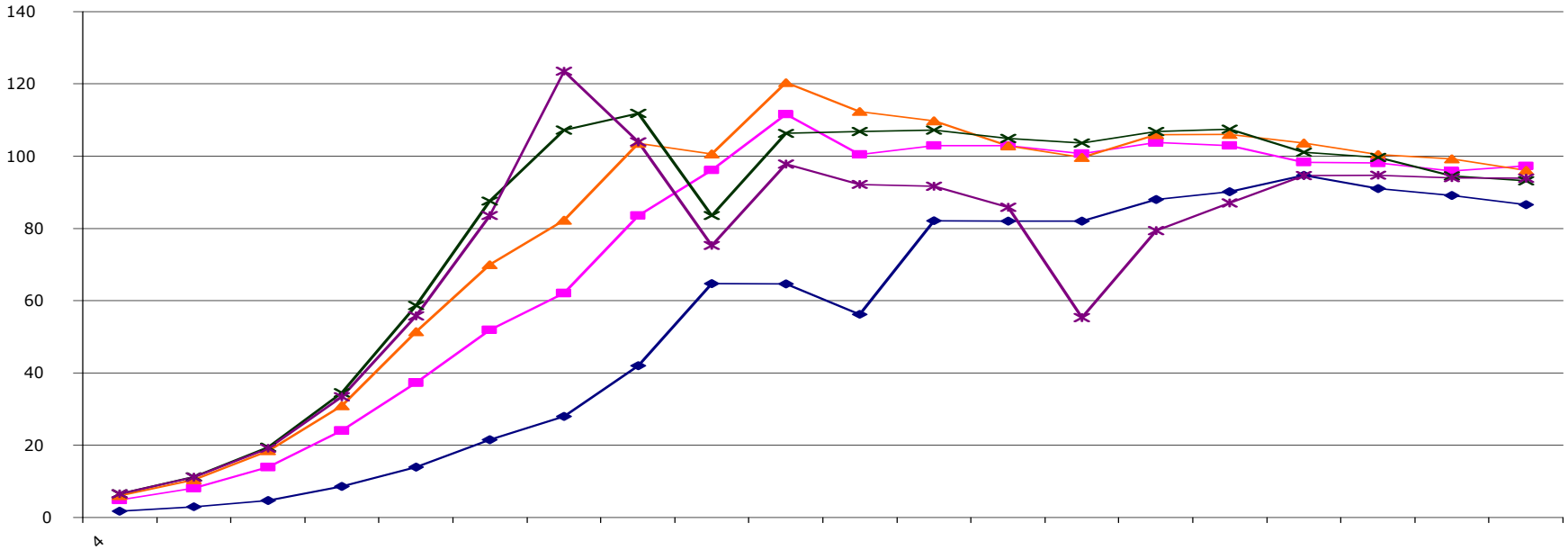



a
b
c
d
0
1
2
3

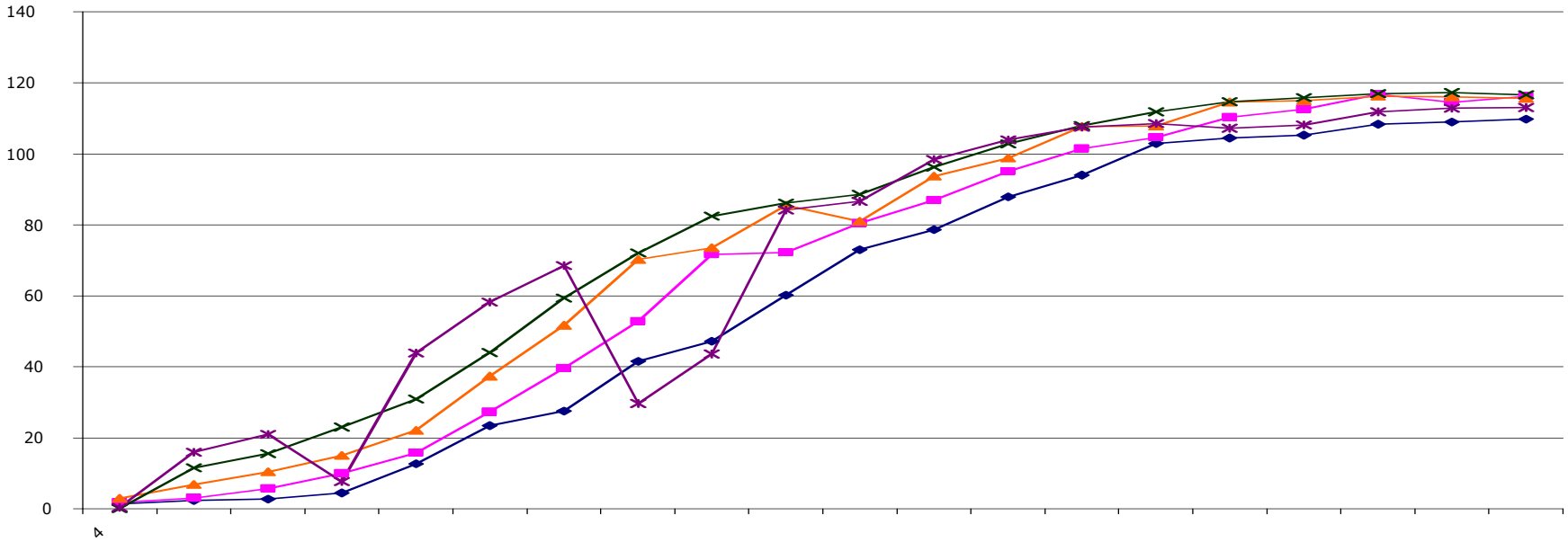

a
a
b
b
c
c
d
d
0
1
2
3


a
b
c
d
a
b
c
d
a
b
c
d
a
b
c
d
0
1
2
3

Alltoall LAM



Alltoall MPJava (prefix algorithm)



NAS PB Conjugate Gradient

Class C Spare Matrix is 150,000 X 150,000
241 nonzero elements per row
36,121,000 total nonzero elements

Class B Sparse Matrix is 75,000 X 75,000
183 nonzero elements per row
13,708,000 total nonzero elements

A . p = q

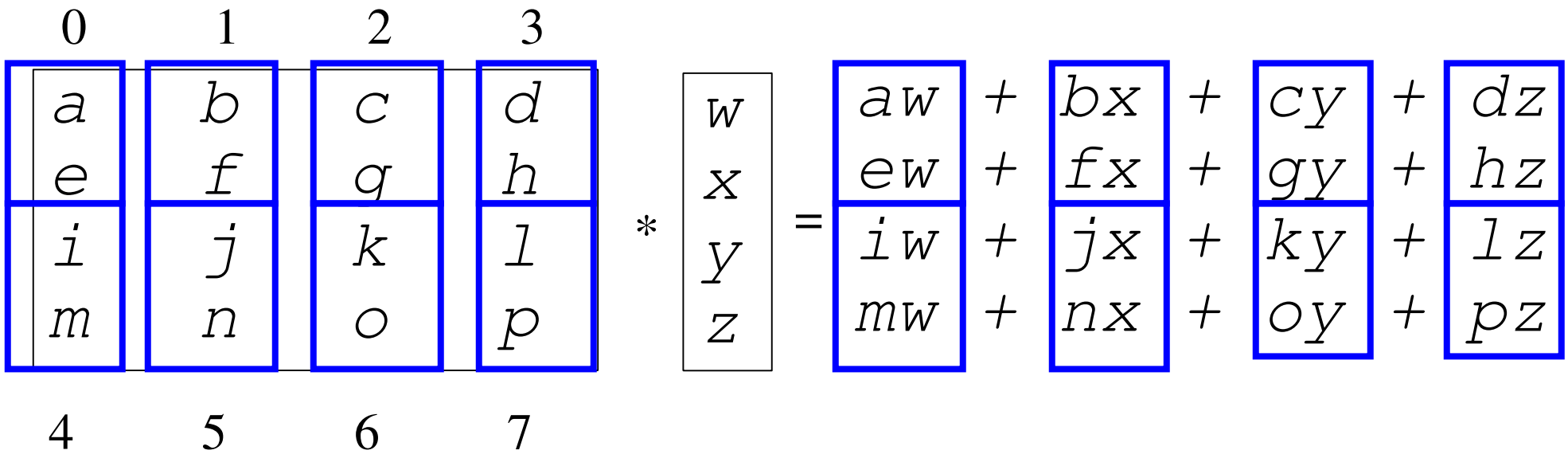
$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} * \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} aw + bx + cy + dz \\ ew + fx + gy + hz \\ iw + jx + ky + lz \\ mw + nx + oy + pz \end{bmatrix}$$

Simple approach to parrallelizing matrix-vector multiple:
Stripe across rows

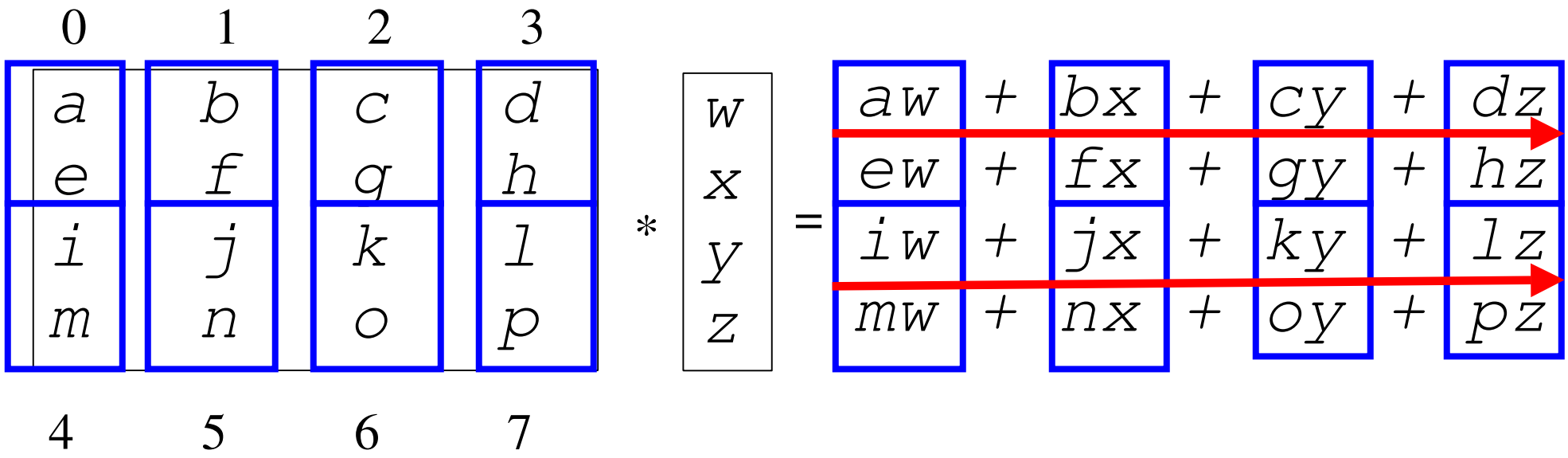
$$\begin{array}{l} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} * \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} aw + bx + cy + dz \\ ew + fx + gy + hz \\ iw + jx + ky + lz \\ mw + nx + oy + pz \end{bmatrix}$$

Requires an all-to-all broadcast to reconstruct the vector *p*

Multi-Dimensional matrix-vector multiply decomposition

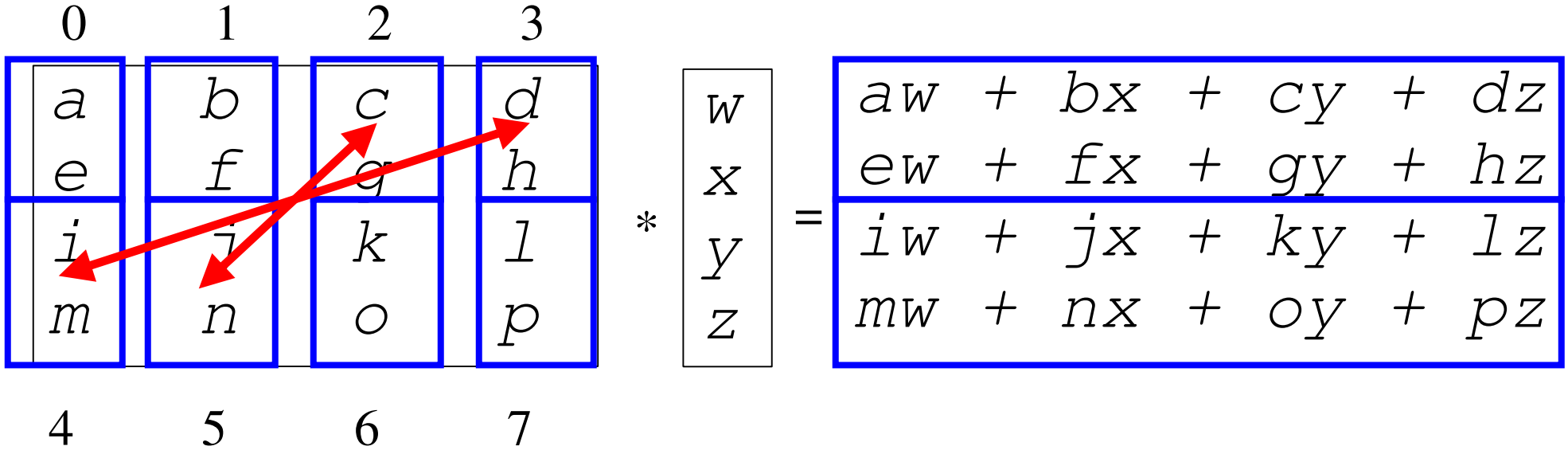


Multi-Dimensional matrix-vector multiply decomposition



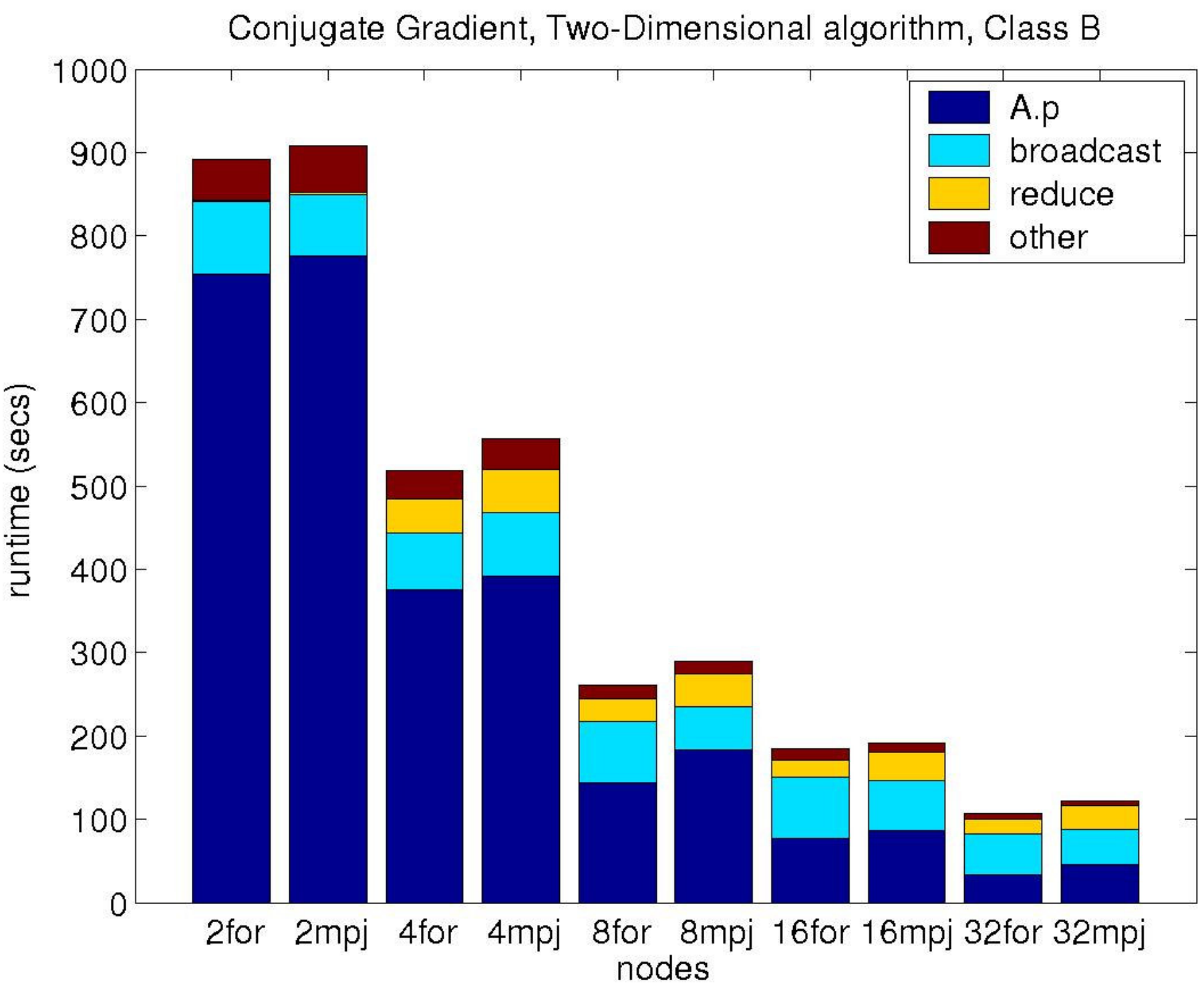
Reduction along decomposed rows

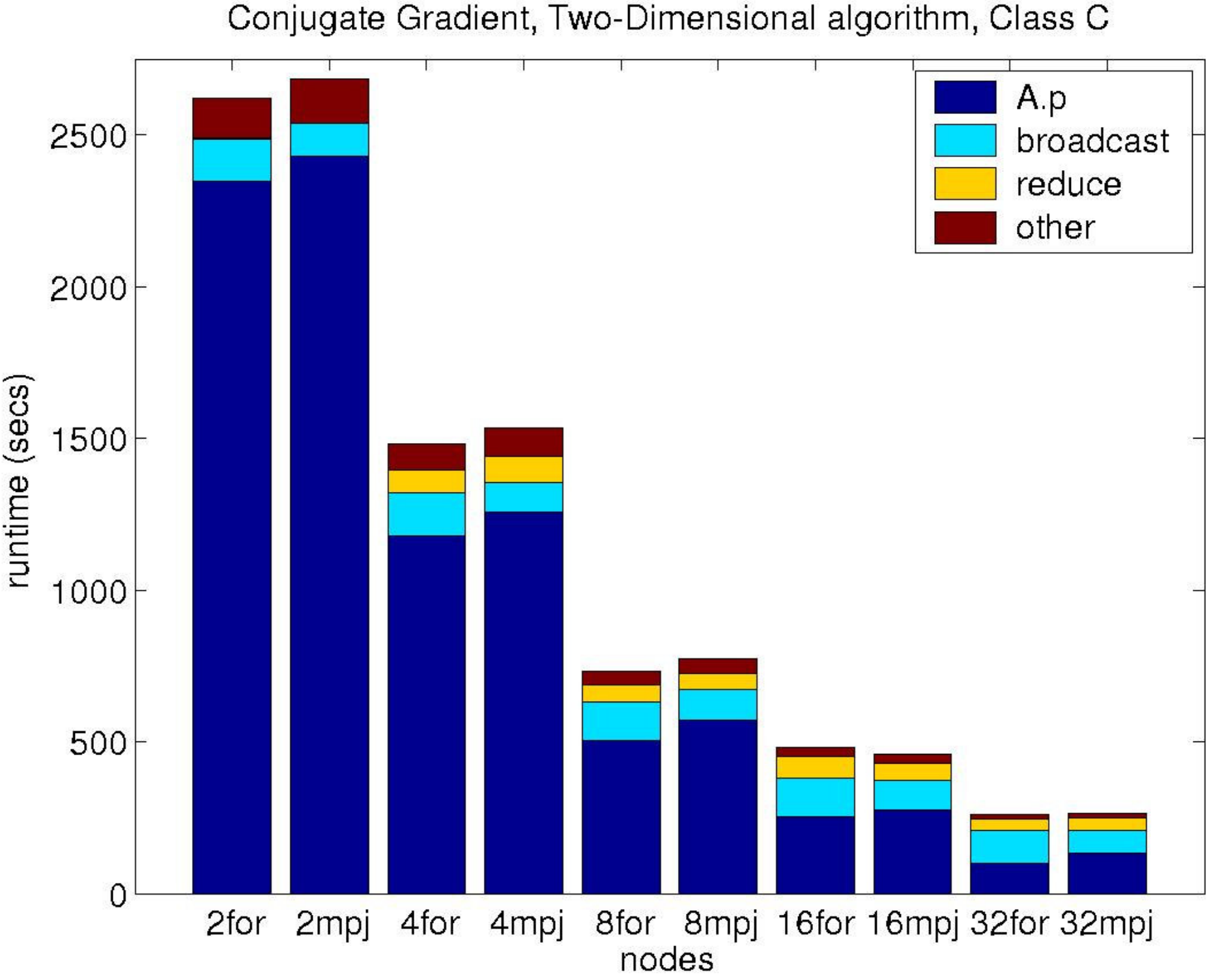
Multi-Dimensional matrix-vector multiply decomposition



Node 4 needs w, and has y,z
Node 3 needs z, has w,x
SWAP

Node 2 needs y, and has w,x
Node 5 needs x, and has y,z
SWAP


Conjugate Gradient, Two-Dimensional algorithm, Class B
runtime (secs)
1000
900
800
700
600
500
400
300
200
100
0
A.p
broadcast
reduce
other
2for
2mpj
4for
4mpj
8for
8mpj
16for
16mpj
32for
32mpj
nodes


Conjugate Gradient, Two-Dimensional algorithm, Class C
A.p
broadcast
reduce
other
runtime (secs)
0
500
1000
1500
2000
2500
2for
2mpj
4for
4mpj
8for
8mpj
16for
16mpj
32for
32mpj
nodes

Conclusion

- A pure-Java message passing framework can provide performance competitive with widely available Fortran and MPI implementations
- java.nio is much faster than the older I/O model
- Java Just-In-Time compilers can deliver competitive performance
- Java has many other useful features
  - type safety
  - bounds checks
  - extensive libraries
  - portable
    - Is performance portable too?
  - easy to integrate with databases, webservers, GRID applications

Future Work

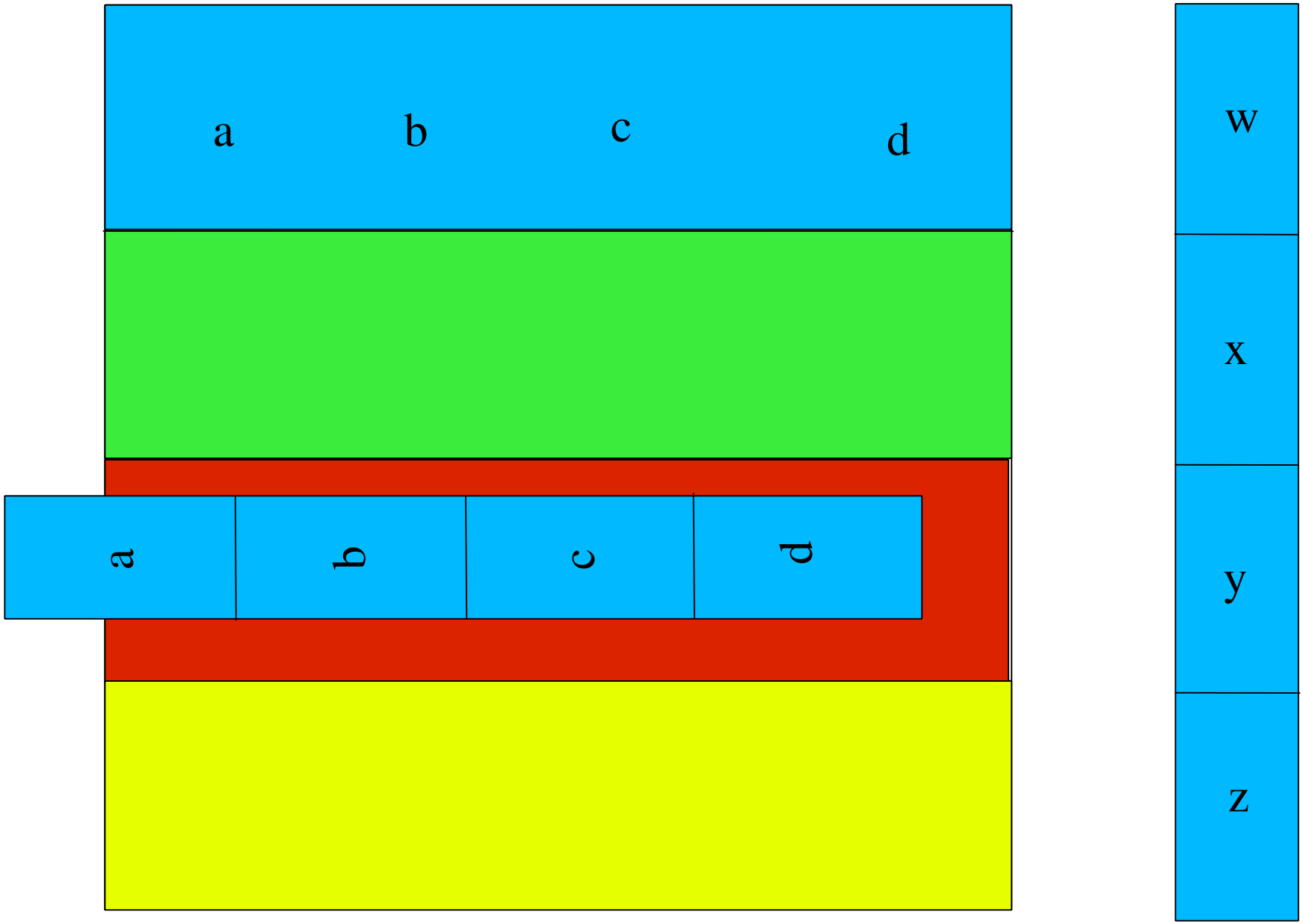
- Exploiting asynchronous pipes
  - Great for work-stealing and work-sharing algorithms, but…
  - subject to Thread scheduling woes
- What about clusters of SMPs?
  - Different bottlenecks
  - More use for multiple threads on a single node
  - Importance of interleaving communication and computation
- Is MPI the right target?
  - BLAS, LAPACK, Netsolver, etc. suggest that programmers will use libraries

Where do we go next?

- Java has the reputation that it's too slow for scientific programming!
  - Is this still accurate?
  - Or were we lucky with our benchmarks?
- Interest in message passing for Java was high a couple of years ago, but has waned
  - Because of performance?
- Does anyone care?
  - Are people happy with Fortran?
  - Is there enough interest in Java for scientific computing?

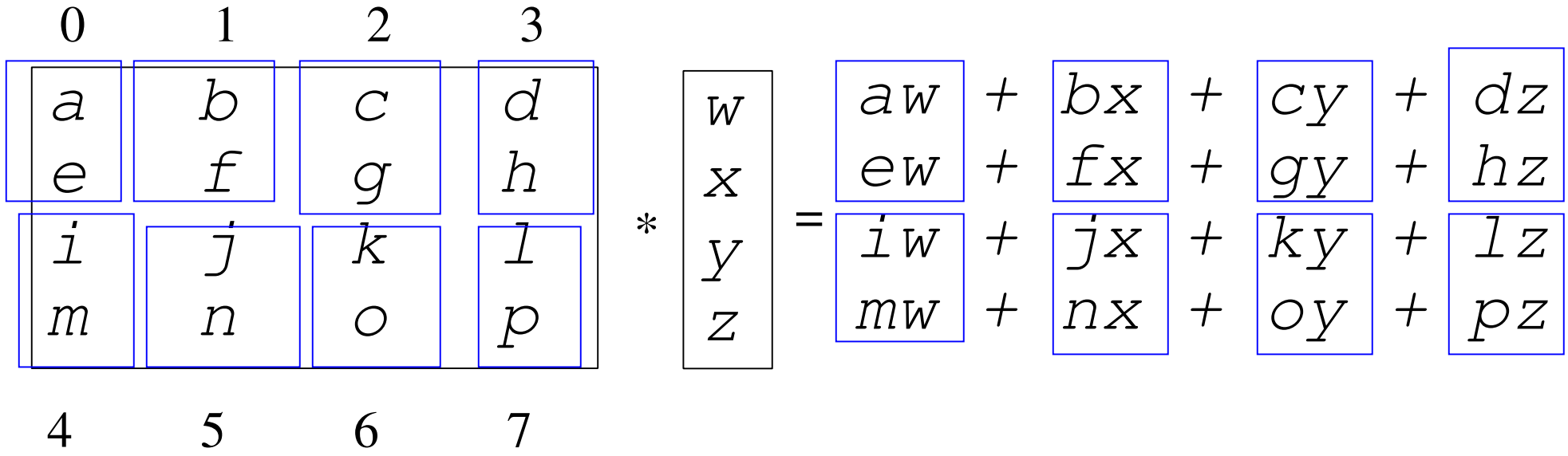
Java may be fast enough but...

- No operator overloading
- No multiarrays package (yet)
  - Also need syntactic sugar to replace .get()/.set() methods with brackets!
- Autoboxing
- Generics (finally available in 1.5)
- Fast, efficient support for a Complex datatype
  - Stack-allocatable objects in general?
- C# provides all/most of these features


a
b
c
d
a
b
c
d
w
x
y
z

NAS PB implementation uses a better algorithm

Multi-Dimensional matrix-vector multiply decomposition



Note the additional swap required for “inner” nodes