

## APIs, Benchmarking & Power of Abstraction

Arkady Kanevsky and Michael Pepe  
Mercury Computers Systems, Inc.

### *Abstract*

This talk examines the abilities of users to develop portable scalable applications without paying the “price of portability.” Three issues will be examined in detail. First, does the API provide the “high enough” level of abstraction for users to develop applications easily? Second, does the API limit implementations from achieving the “optimal” performance by not allowing the use of platform architecture features to the fullest? Also do implementations use the full power of the architectural features? Finally, do the benchmarks fully evaluate the capabilities of the middleware products and allow them to demonstrate the platform architectural abilities fully? All these questions will be demonstrated on the Mercury platform with RACE MPI/RT on distributed matrix transpose (cornerturn) benchmark.

MPI/RT-1.0 was ratified in December 1998 and the final standard version was approved in March 2000, after many corrections based on feedback from the implementers and users. MPI/RT provides a channel level abstraction for message transfer management, buffer abstraction for message memory management, publish-subscribe events abstraction for control flow management, group abstraction for process set management, and various supporting functionality. MPI/RT-1.0 is OO-based, defines production and development libraries, and is targeted for real-time and high-performance parallel computing applications. The on-going MPI/RT-1.1 forum marginally extends the above abstractions to provide more flexibility to the users and to allow implementations to utilize platform resource more efficiently and to provide users higher performance. All extensions preserve the frame of the above abstractions.

The in-place distributed 2D matrix transpose benchmark takes a two-dimensional matrix distributed over a set of processes stored in row order and reorganizes it so the matrix is stored in column order. This is a typical step between two processing stages to achieve lower latency for signal and image applications. MITRE had developed a set of public portable MPI-based benchmarks that included the cornerturn one. Several groups of researchers independently converted that benchmark into an MPI/RT-based one. On the Mercury platform, the MPI/RT-based cornerturn provided lower latencies than the MPI-based one but the price of portability was still high.

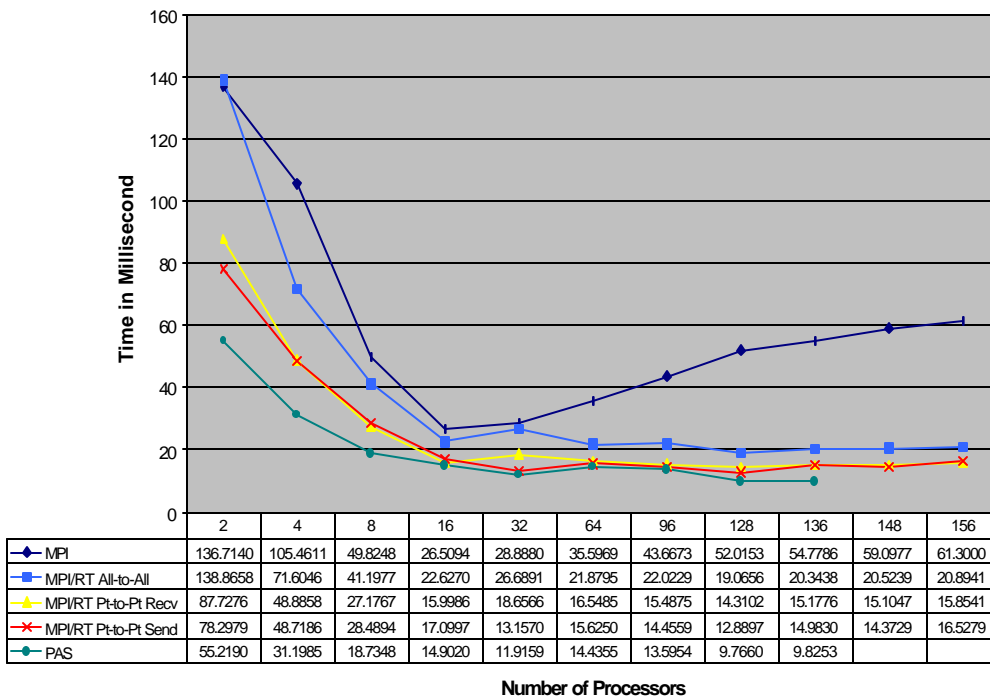
Examining the reasons for it, several things become apparent. First, the cornerturn is such a basic block of communication that it should be brought to the API level. Data Reorganization (DR) standard is already incorporating it into the API to provide a higher level of abstraction for users. This is trickier than it sounds because the cornerurn benchmark requires not only communication between processes that store the matrix but also the local submatrix transpositions that involve computation. Second, the portable benchmark is designed in such a way that it does not allow users to overlap local computation with global communication, hence not allowing users to take full advantage of the platform architecture. Most of the high-performance multicomputers have a separate communication engine independent from the CPU. Mercury has a DMA engine that moves data independently from the CPU. This allows users to overlap computation and communication and hide the time for one in the time of another.

In order to take advantage of computation/communication overlap, the cornerturn benchmark must be rewritten. But can it still be portable? Can it scale with matrix sizes? With the number of processors? What if the processor speed or DMA transfer rate changes? What will happen on a different architecture? This and other issues will be addressed in detail during the presentation. Other issues that affect performance of the benchmark consist of who starts the DMA transfer (sender or receiver), alignment of the messages, and granularity of synchronization (single message vs multiple messages). Below we provide an evolution of the cornerturn benchmark results.

During the cornerturn benchmark redesign, one shortcoming of the MPI/RT-1.0 API has become apparent. It deals with the memory management abstraction. The current buffer specification requires that a buffer consists of one contiguous block of memory. On the one hand, it does not allow the user to transfer a submatrix from one processor to another without first copying it into or from a contiguous memory buffer. On the other hand, it does not allow the implementation to take advantage of chained DMA that is available on several platforms including Mercury RACE and RACE++.

All of these issues point out the need of a higher level abstraction to ease the application writer's job but will still allow middleware implementers to provide "optimal" performance. That API should provide complete performance power to users of platform architectural features and should allow implementers to use the best features that the platform has to offer. Many of the issues of communication and computation overlap have to be revisited at a higher level when users take a look at the application as a whole rather than in individual stages. Issues of hiding resources but providing abstraction of them to the users are central to the API design. The simultaneous push from the bottom up as in the case of MPI/RT and DR, and from the top-bottom as in the case of GADEI & RT-Express is helping to hash many of these issues, but this process is far from being done. Some of the details of DR interface will be provided to demonstrate the power of abstraction.

1024 x 1024 Cornerturn 375 MHz PPC750s





# APIs, Benchmarking & Power of Abstraction

Arkady Kanevsky and Michael Pepe  
Mercury Computers Systems, Inc.



# Portable Scalable Benchmarks

- How to develop portable scalable benchmarks?
  - » Portability vs. performance
- Benchmarks evaluate platforms:
  - » Hardware (processors and interconnects)
  - » OS
  - » Middleware
- Benchmarks use standard or standard-based middleware to achieve portability



# Middleware APIs

- API issues to consider
  - » Does API provide a high enough level of abstraction to hide platform-specific details?
  - » Does API limit implementations from achieving performance by restricting use of platform-specific architectural features?
  - » Does API implementation use that power?
- Does benchmark use middleware properly?



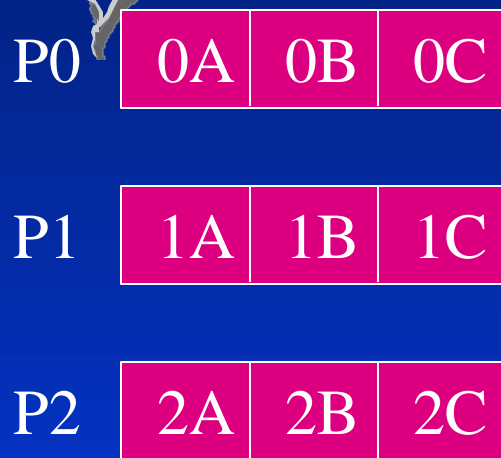
# 2D Cornerturn Example

- MPI-based MITRE portable benchmark
  - » Local Cornerturn - computation
  - » Data Redistribution - communication
  - » Local Reorganization - computation
- Does it really measure the platform capabilities?
  - » It was modified for specific platforms to increase performance. Still MPI-based and portable but what about performance on different platforms?

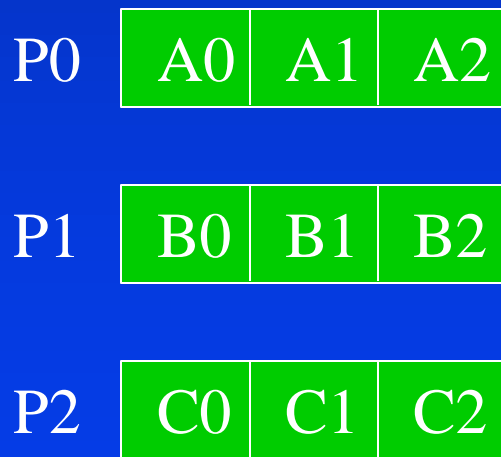
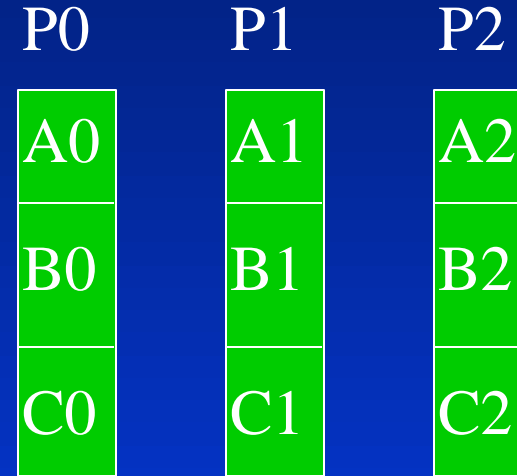


# 2D Cornerturn Example

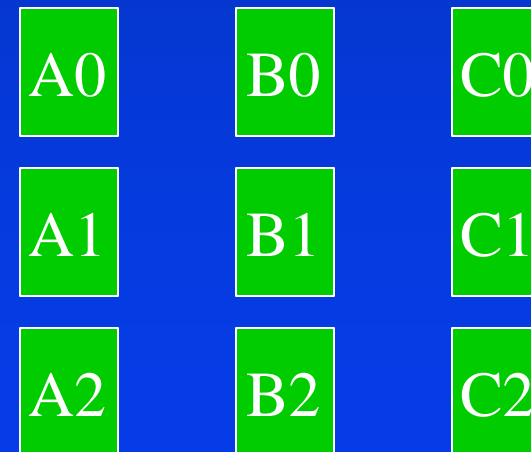
(Continued)



Local  
Cornerturn  
→



Local  
Reorg  
←



Redistributon

# 2D Cornerturn Example

(Continued)



- Converted to MPI/RT version by Mercury without changing logic
  - » NO overlap of computation and communication
    - Benchmark design issue
    - Does NOT measure platform capabilities
- Cornerturn should be brought to the API level
  - » Users need to know platform-specific architectural features and redesign this fundamental application block from one platform to another to get performance
  - » DRI takes care of it!
- Does application use cornerturn block as is or is it possible to overlap it with application computation?

# 2D Cornerturn Example

(Continued)

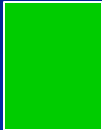
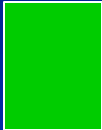
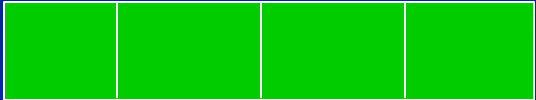
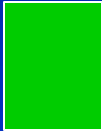
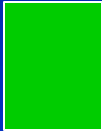
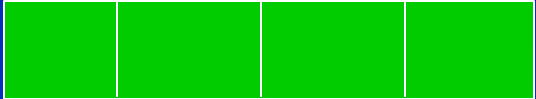
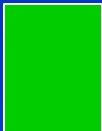
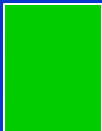
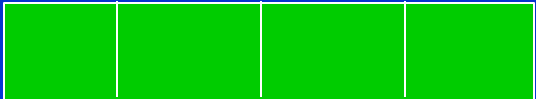


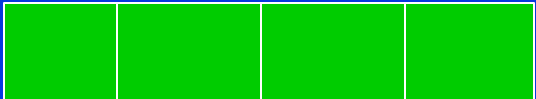


- Convert ALL-TO-ALL based communication to individual processor-to-processor transfers with overlapping transfer and local processing
  - » Processor K: for ( $J=K+1 \pmod N$ ;  $J < K+N-1 \pmod N$ ,  $J++$ )
    - Local cornerturn for  $J+2$ -th processor (next iteration)
    - Send cornerturned data to  $J+1$ -st processor (current iteration)
    - Local reorg data received from  $K-J \pmod N$  (previous iteration)



# 2D Cornerturn Example

(Continued)

	INPUT	Send	Receive	OUTPUT
P0	0A 0B 0C 0D			
P1	1A 1B 1C 1D			
P2	2A 2B 2C 2D			
P3	3A 3B 3C 3D			



# 2D Cornerturn Example

(Continued)

	INPUT	Send	Receive	OUTPUT
P0	0A   0B   0C   0D	B0		
P1	1A   1B   1C   1D	C1		
P2	2A   2B   2C   2D	D2		
P3	3A   3B   3C   3D	A3		



# 2D Cornerturn Example

(Continued)

	INPUT	Send	Receive	OUTPUT
P0	0A 0B 0C 0D	C0	A3	
P1	1A 1B 1C 1D	D1	B0	
P2	2A 2B 2C 2D	A2	C1	
P3	3A 3B 3C 3D	B3	D2	



# 2D Cornerturn Example

(Continued)

	INPUT	Send	Receive	OUTPUT
P0	0A   0B   0C   0D	D0	A2	A3
P1	1A   1B   1C   1D	A1	B3	B0
P2	2A   2B   2C   2D	B2	C0	C1
P3	3A   3B   3C   3D	C3	D1	D2



# 2D Cornerturn Example

(Continued)

	INPUT	Send	Receive	OUTPUT
P0	0A   0B   0C   0D		A1	A0     A2   A3
P1	1A   1B   1C   1D		B2	B0   B1     B3
P2	2A   2B   2C   2D		C3	C0   C1   C2
P3	3A   3B   3C   3D		D0	D1   D2   D3



# 2D Cornerturn Example

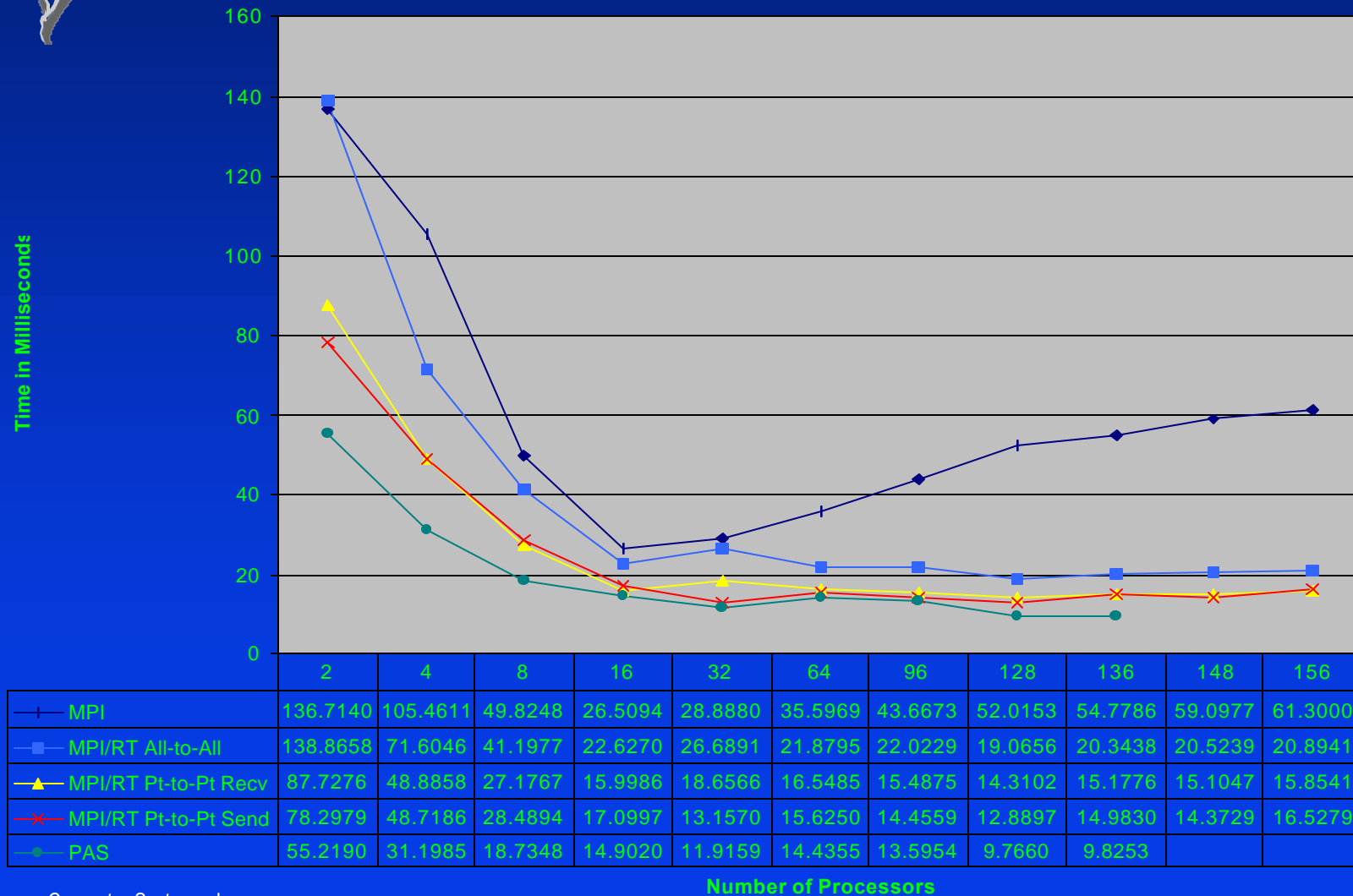
(Continued)

	INPUT	Send	Received	OUTPUT
P0	0A   0B   0C   0D	■	■	A0   A1   A2   A3
P1	1A   1B   1C   1D	■	■	B0   B1   B2   B3
P2	2A   2B   2C   2D	■	■	C0   C1   C2   C3
P3	3A   3B   3C   3D	■	■	D0   D1   D2   D3



# 2D Cornerturn Benchmark Progression

1024 x 1024 Cornerturn 375 MHz PPC750s





# 2D Cornerturn Example

(Continued)

- Is the new benchmark version portable?
  - » Yes, it uses MPI/RT calls
- Is it performance portable?
  - » Not always; yes if
    - Architecture supports computation and communication overlap and
    - MPI/RT implementation takes advantage of it
- Is the benchmark still scalable?
  - » With matrix sizes - Yes
  - » With number of processors - Yes

# 2D Cornerturn Example

(Continued)



- Is this benchmark design still valid if processor or network speed changes?
  - » Benchmark will still work
  - » Performance is no longer optimized
  - » Ratio between computation-communication speeds is main factor in scalability decision
- Other issues to consider
  - » Message alignment (for computation, communication)
  - » Sender/receiver does message transfers
  - » Granularity of synchronization between processors

# 2D Cornerturn Example

(Continued)



- Cornerturn benchmark design issues
  - » Why do we need a receive buffer?
  - » Can we receive directly into an output buffer?
    - This requires receiving data into a non-contiguous memory buffer, that is, into a strided memory buffer
  - » Symmetrically for the send side
- Can platform (hardware and middleware) support strided buffers for send and/or receive?
  - » Mercury hardware supports strided buffers for send side
  - » MPI/RT and MPI do not support strided buffers for send or receive sides - API deficiency
  - » PAS supports strided buffers



# Conclusion

- Need higher level abstraction for middleware APIs
  - » DRI addresses this issue
  - » Parallel VSIPL?
- Benchmarks should be designed in such a way that platform-specific features can be exploited without redesign
- Use benchmarks results properly!
  - » How does the benchmark fit into an application
    - Resource usage in parallel (overlap of computation and communication)
  - » Platform for multiple benchmarks for an application is the same