Compiler-based Autotuning of MPI

Martin Swany

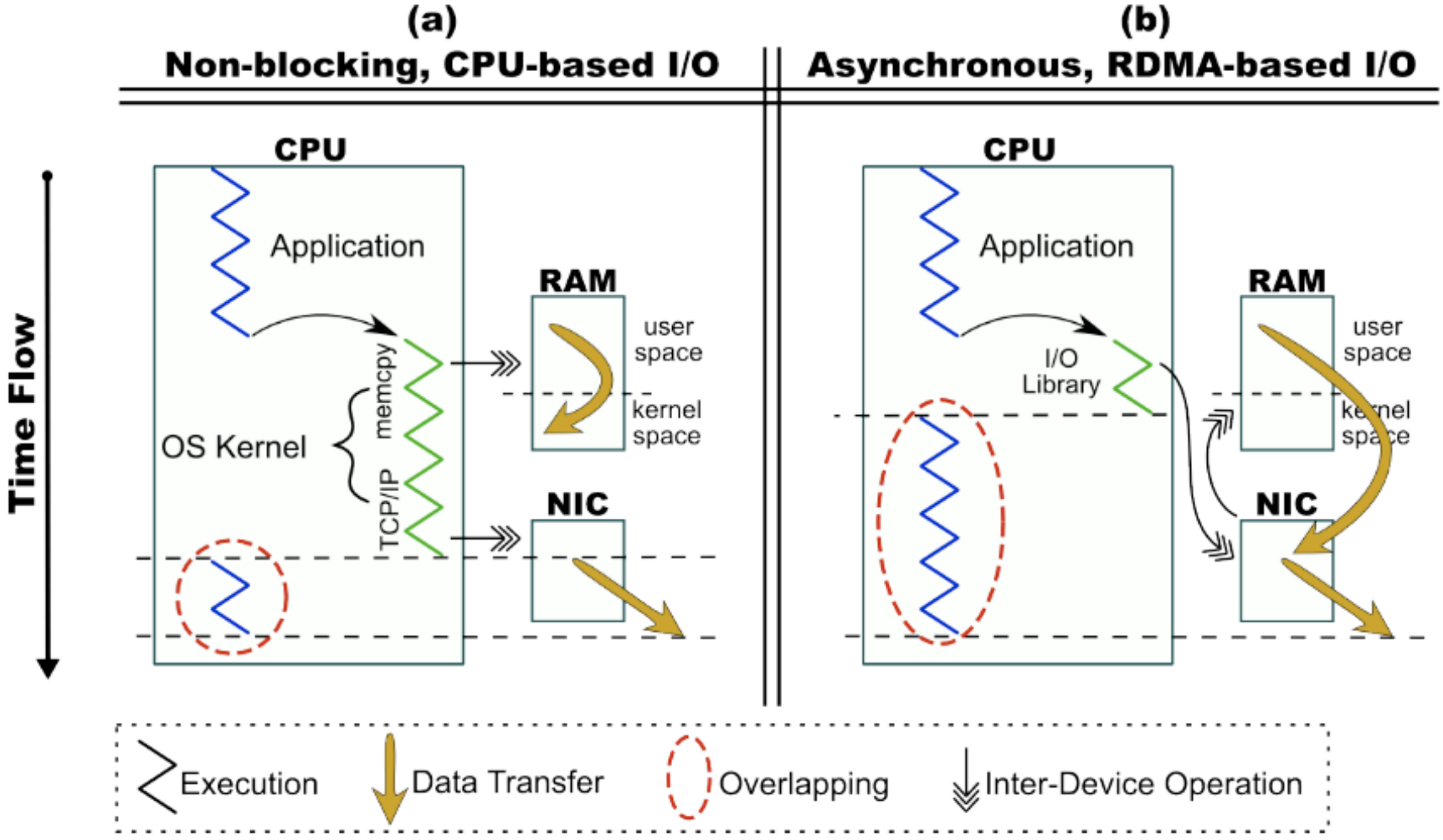
Application Transformation

- Parallel application performance depends on efficient data movement
- Programming methodologies for good message-passing performance can be difficult to program and maintain
  - Asynchronous and one-sided messaging
  - Specifics of network interfaces change
- This work focuses on automatic program transformations to reduce the overhead of communication (and programmer effort)
- This approach will be critical for petascale systems!

Autotuning Position

- Question: Suppose all layers of the software stack (e.g., OS, middleware, MPI, libraries, apps) are "autotuned." Will we need to integrate these multiple layers, and if so, how?
- Position: There are certainly interactions between autotuned libraries. Compute kernels and message-passing code should be tuned together.

Overlapping Computation and Communication



Overlapping Details

- Minimize overhead of data movement by overlapping it with useful work
  - An well-known idea
- What does it mean for parallel application structure?
  - Post a send as soon as the data is ready (without copying, if possible)
  - Do useful work
  - Check status after completion (with minimal polling, sleeping or busy-waiting)
- Difficult to optimize, difficult to maintain
  - Not portable across platforms

Basic Approaches

- Compiler-based application transformation
  - Previously only source-to-source, now to binary
- Transform MPI communication
  - Collectives → Point-to-point
  - Blocking → Non-blocking
  - Non-blocking → One-sided
  - Send fission and fusion
    - Strip-mining
- Separate costs of communication
  - Hoist
  - Overlap

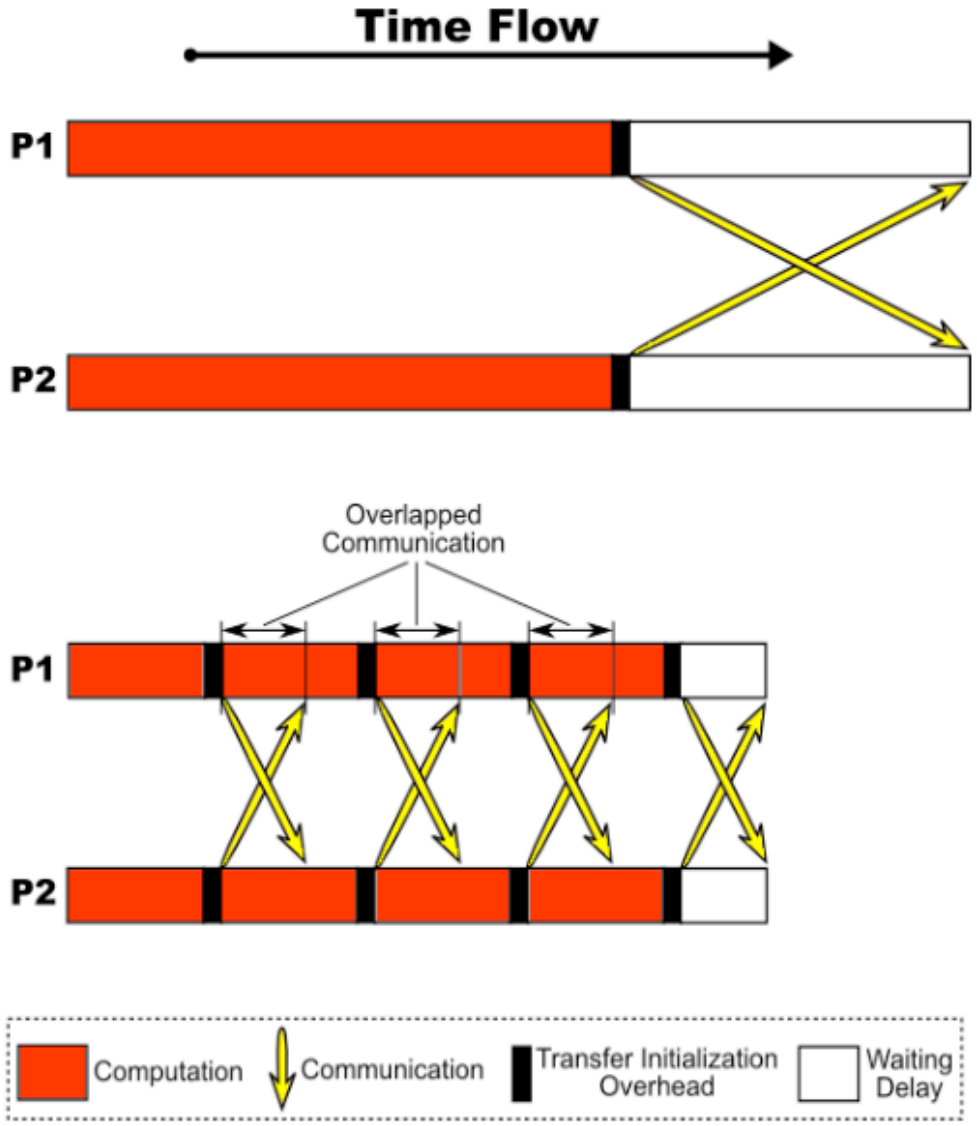
Overlapping Transformation - Simple Example

Original code

```
integer, dimension(M,N):: array

do i = 1, N
   /* computation kernel */
   subroutine( array(1,i) )
enddo

size = M*N
DataTransferCall( array(1,1), size, ... )

Other_Computation()
```

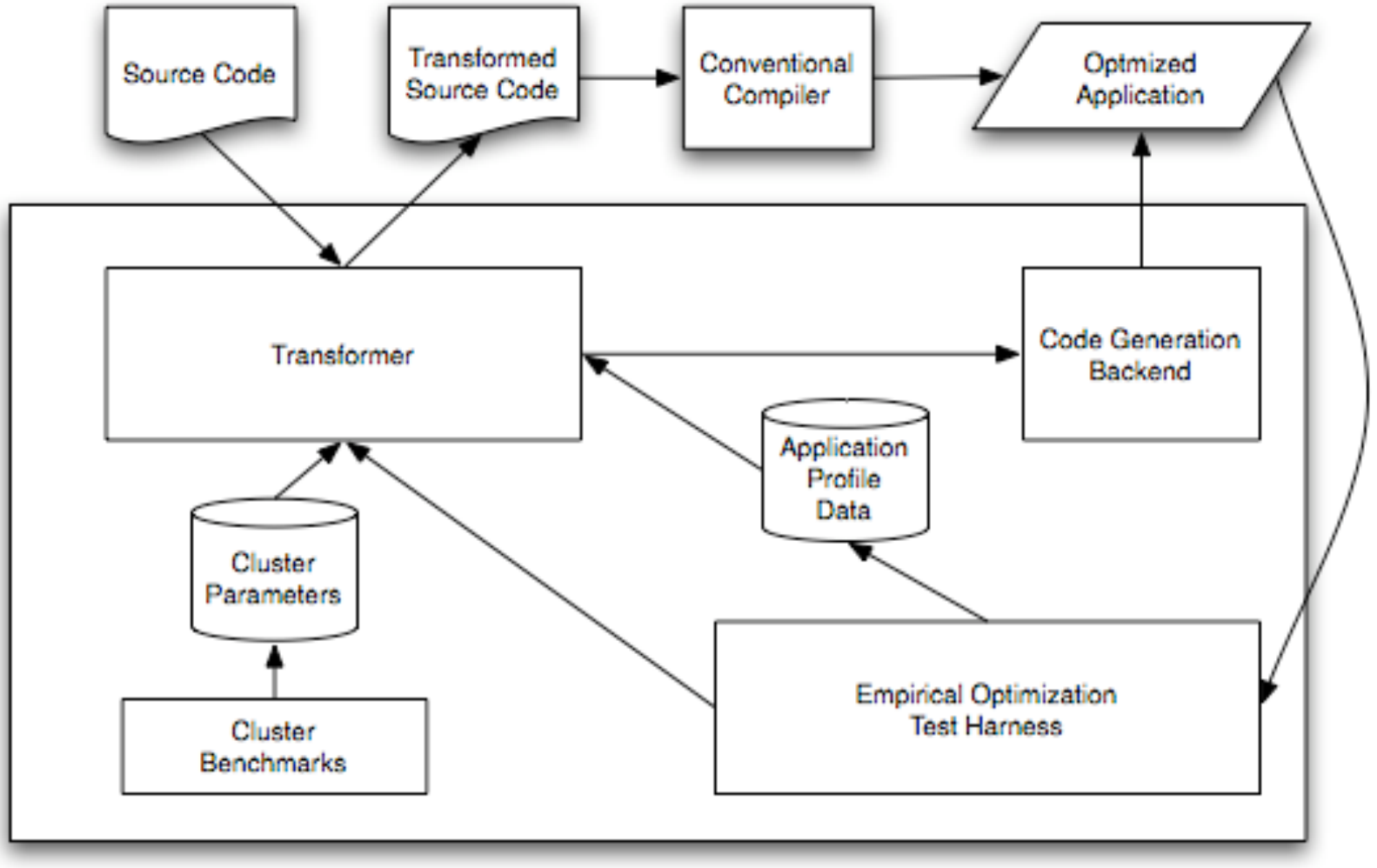
Tiled code

```
integer, dimension(M,N):: array

do i = 1, N, K

   do j = i, i+K-1
      /* computation kernel */
      subroutine( array(1,j) )
   enddo
   if( i > K ) then
      /* block for the arrival of the data  */
      MPI_WAITALL( request( i - K) )
   endif

   size = M*K
   /* asynchronous network transfer */
   MPI_ISEND( array(1,i), size, ... )
   MPI_IRECV( destn(...), request(i), ... )
enddo

MPI_WAITALL( request( i - K) )
```

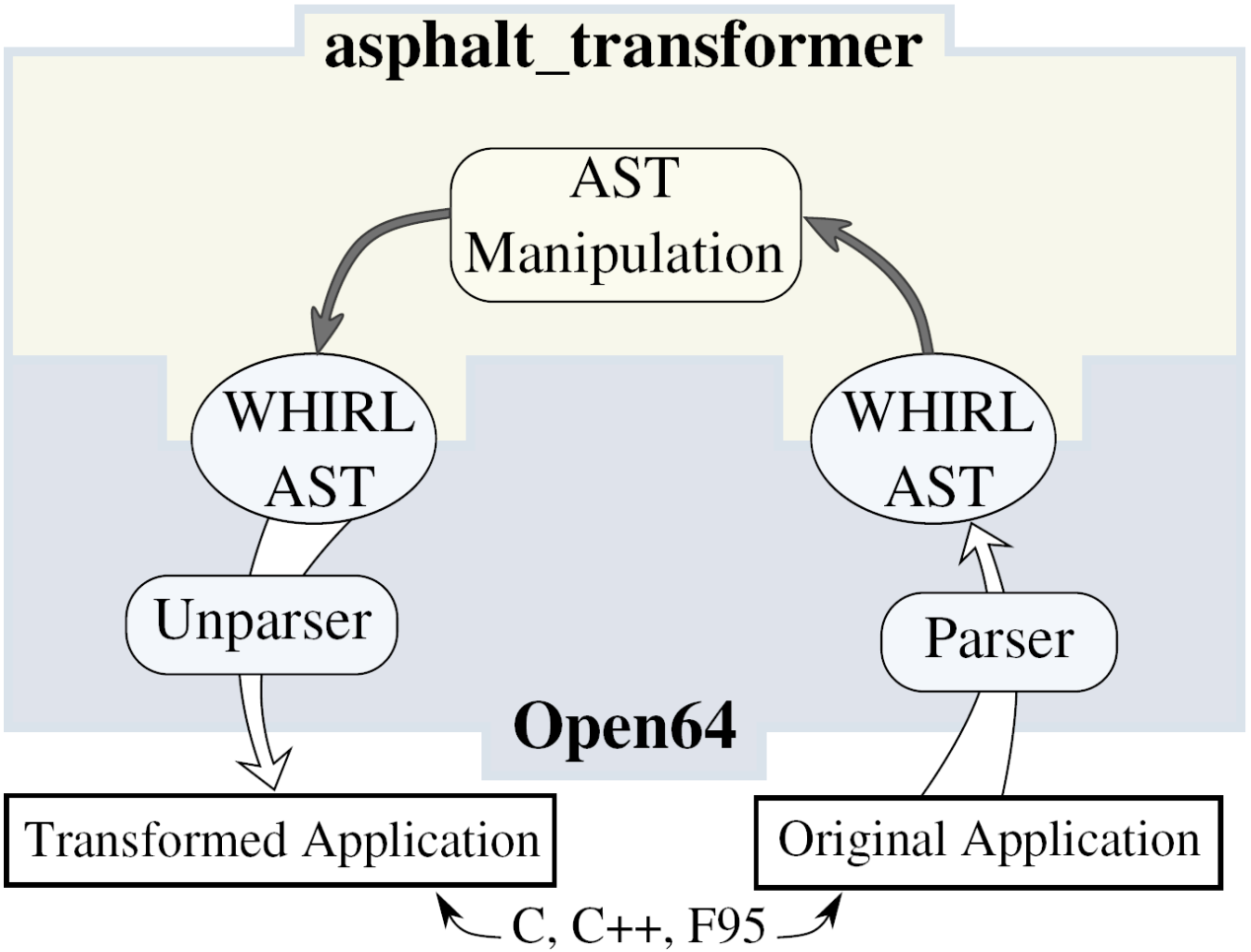


ASPhALT

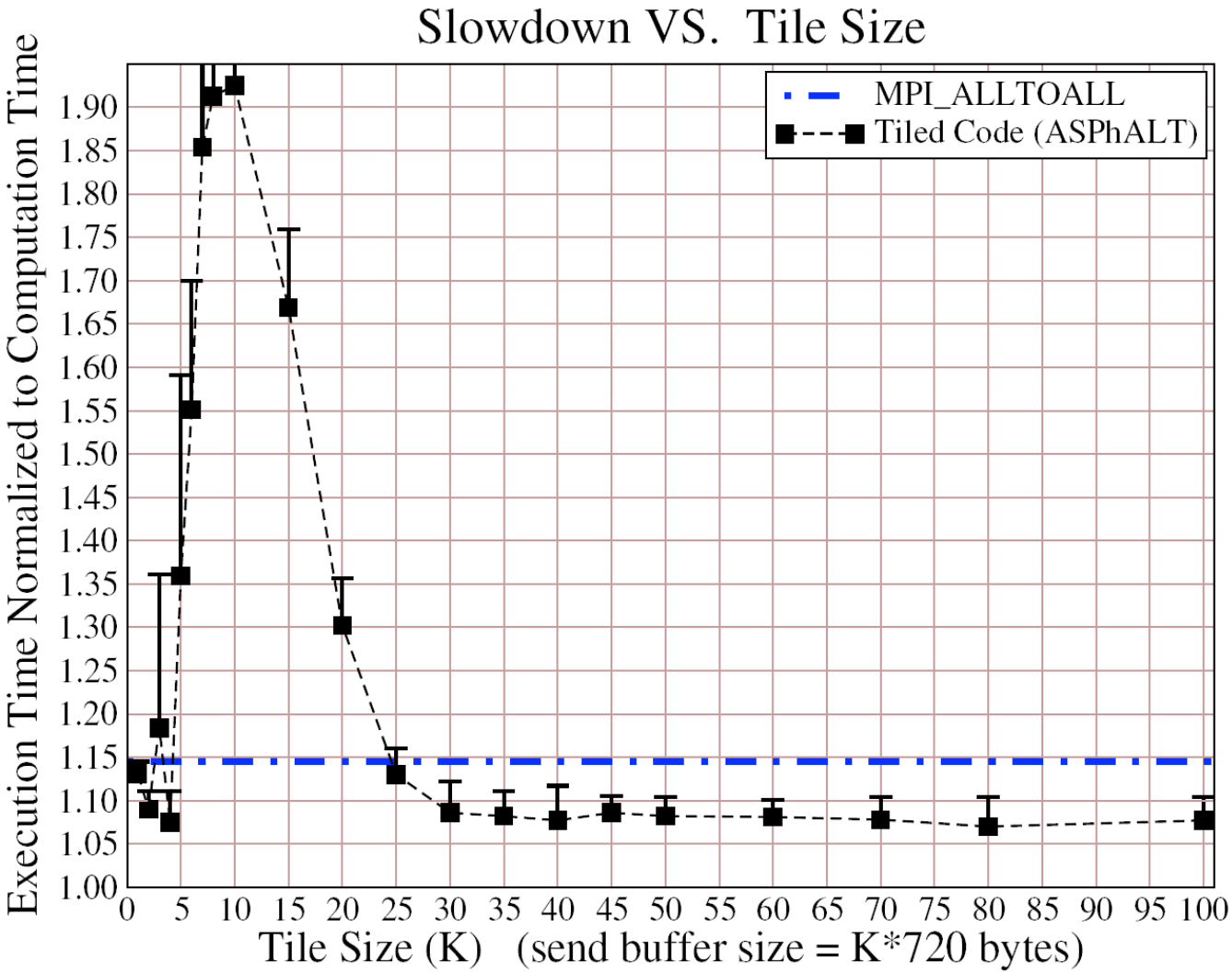
- **A**utomatic **S**ystem for **P**arallel **A**pp**L**ication **T**ransformation



Transformer Structure

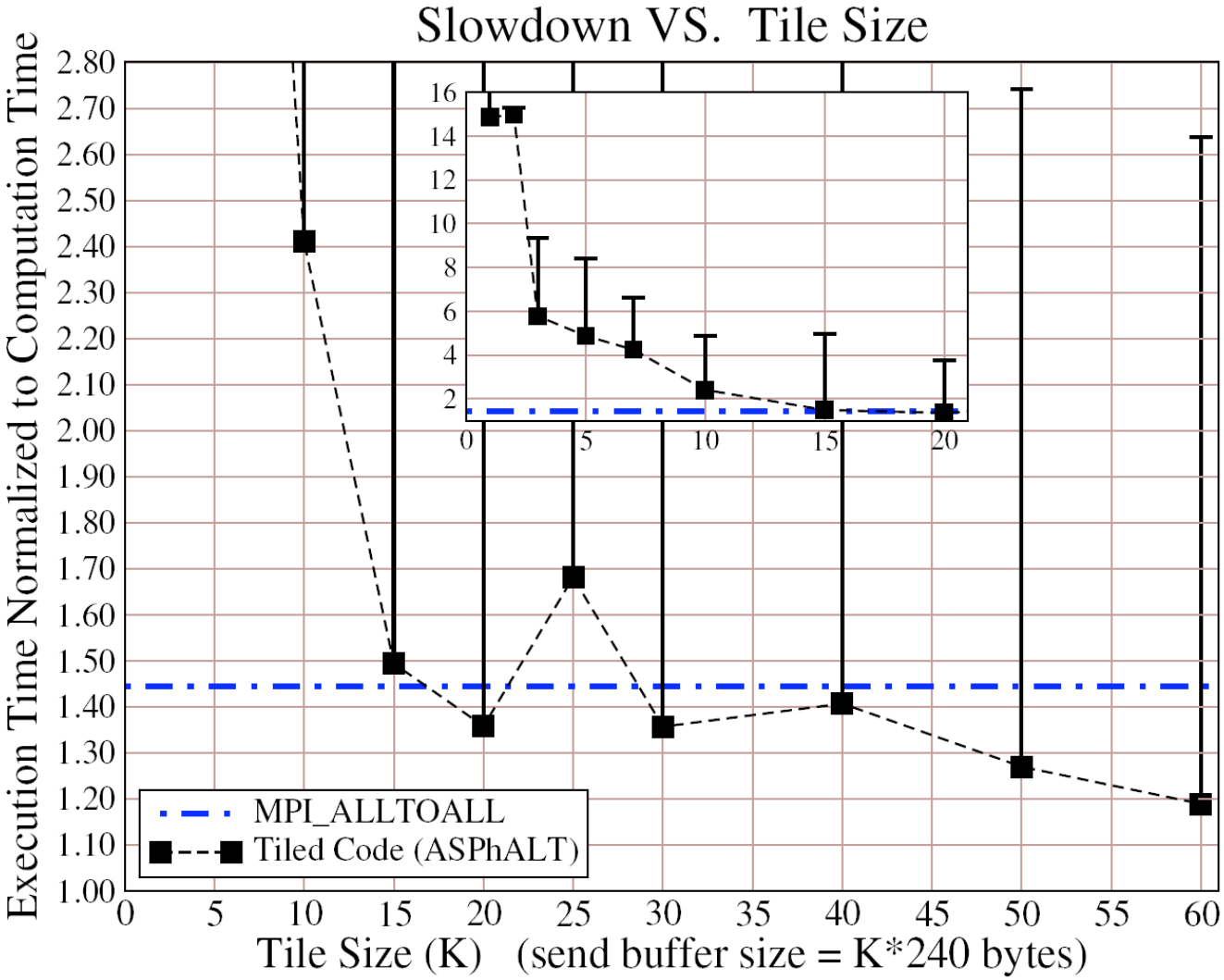


Evaluation of Automatic Transformation - Synthetic Kernel



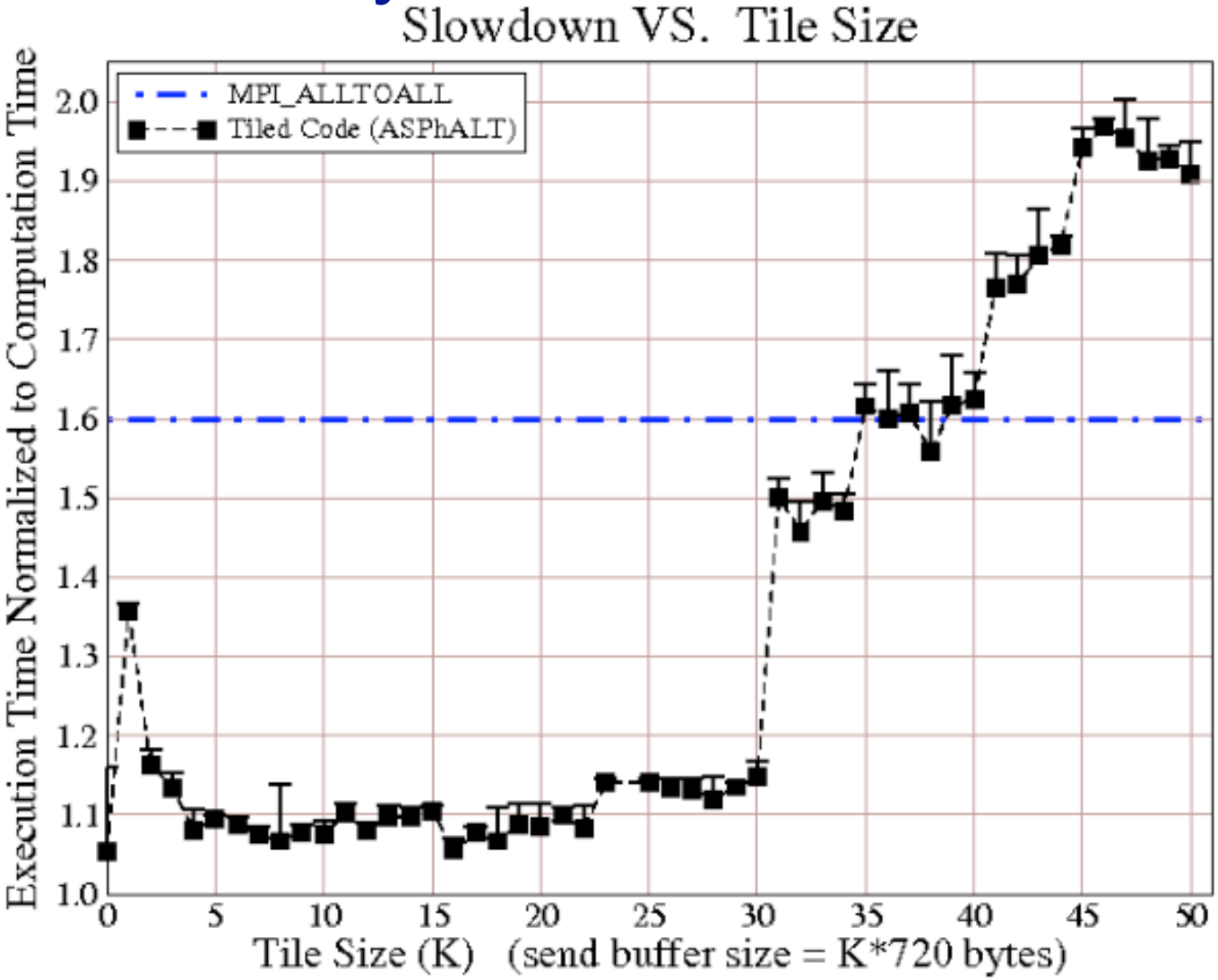
interconnect:**Ammasso**, NP:**16**, size:**1440x1440x48x16** Bytes

Evaluation of Automatic Transformation - Synthetic Kernel



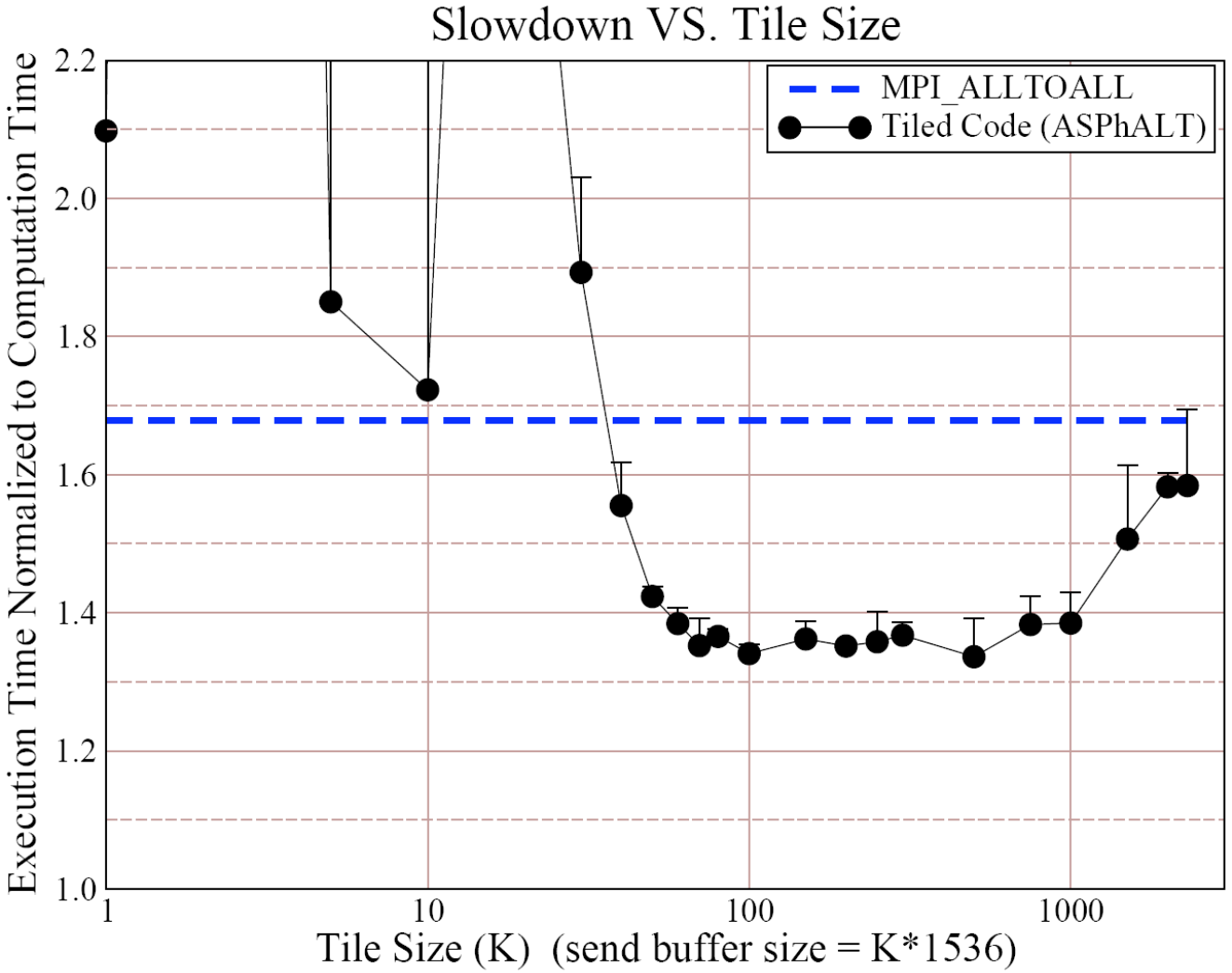
interconnect:**Myrinet-MX**, NP:**48**, size:**1440x1440x48x16** Bytes

Evaluation of Automatic Transformation - Synthetic Kernel



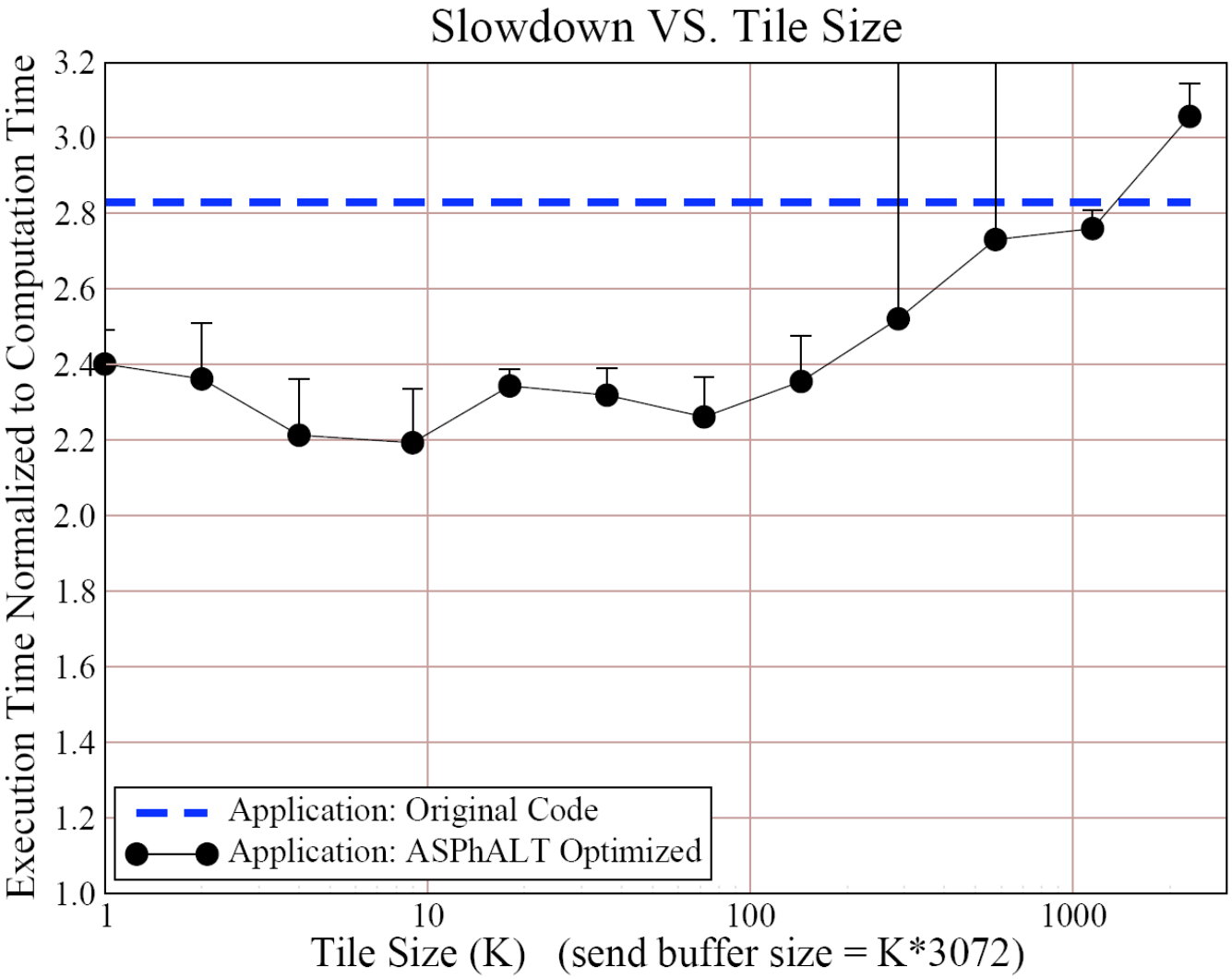
interconnect:**SCI from Dolphin**, NP:**8**, size:**1440x1440x48x16** Bytes

Evaluation of Automatic Transformation - Application “visco”



interconnect:**Myrinet-MX**, NP:**48**, size:**9216x2305x48x16** Bytes

Evaluation of Automatic Transformation - Application “visco”

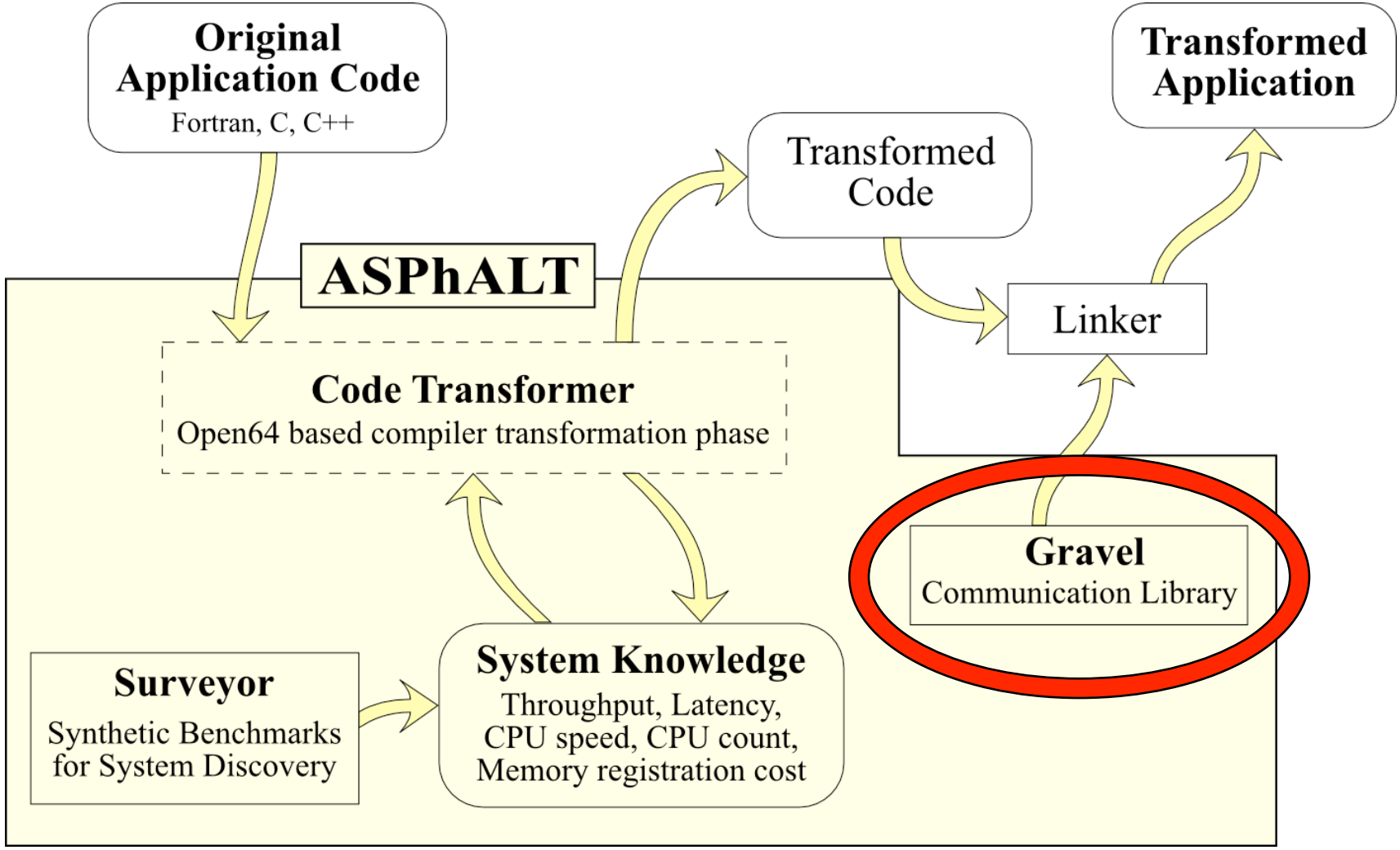


interconnect:**Myrinet-GM**, NP:**24**, size:**9216x2305x48x16** Bytes

Autotuning of Tile size

- The tile size is an obvious choice for autotuning
  - Though not covered here, another parameter we have investigated is how many tiles should be outstanding in the pipeline
- These results were for MPI_ALLTOALL but other work has considered single send/recv pairs and scatter /gather
  - Matching done with pragma
- Clearly there is an interaction between our transformation and the loop transformations performed for compute kernels
- We're limiting ourselves if we have an optimized compute phase followed by an optimized communication phase!

ASPhALT and Gravel



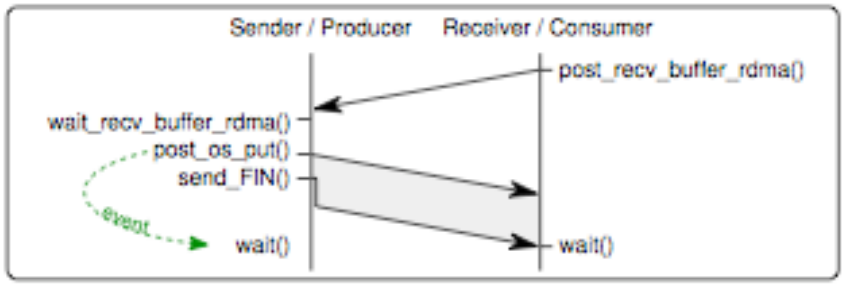
Gravel – An MPI Companion Library

- Decompose messaging components
  - Memory registration (for DMA)
  - Message metadata, or header
    - Rendezvous or handshake if no message buffering
  - Message data
- Implement a lightweight system library atop uDAPL from OpenFabrics
  - Possibly still too high level
- Build up abstractions that facilitate replacement of performance-critical MPI calls
  - Not a replacement for MPI

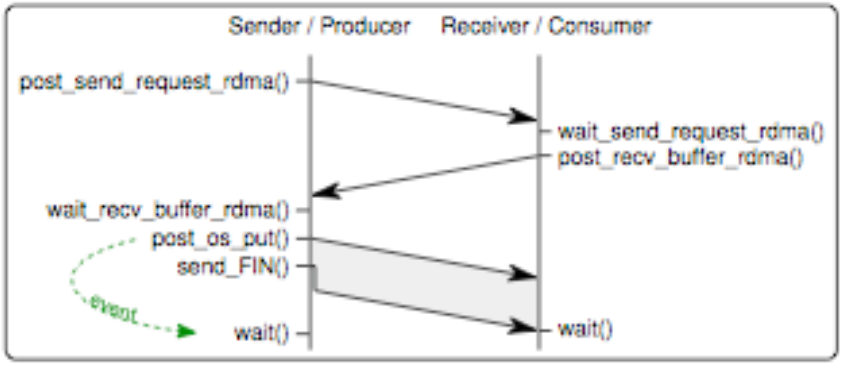
Gravel

- Explicit memory registration
  - Rather than custom memory allocator
- No message buffering
  - No unexpected message queue
  - No “eager” mode
- Message metadata and completion indication also use RDMA to specific locations in peer memory called “ledgers”
  - Can enable true overlap

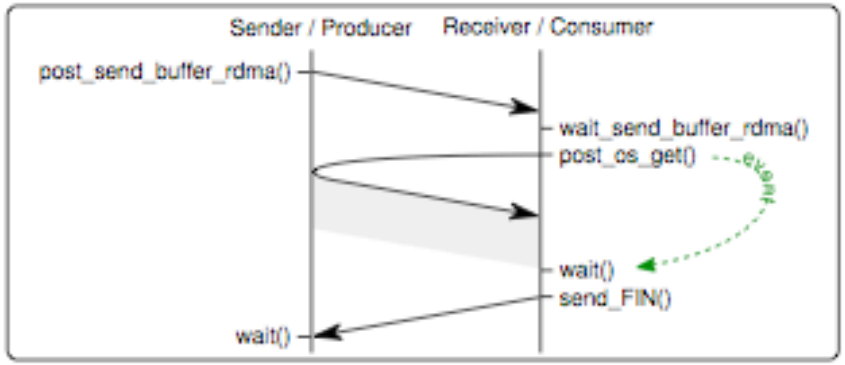
Gravel Rendezvous Protocols



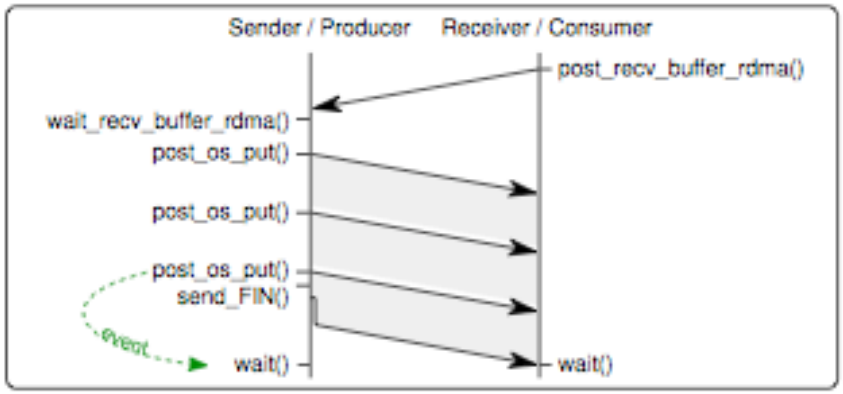
(a) Consumer Initiated RDMA write protocol



(b) Producer Initiated RDMA write protocol

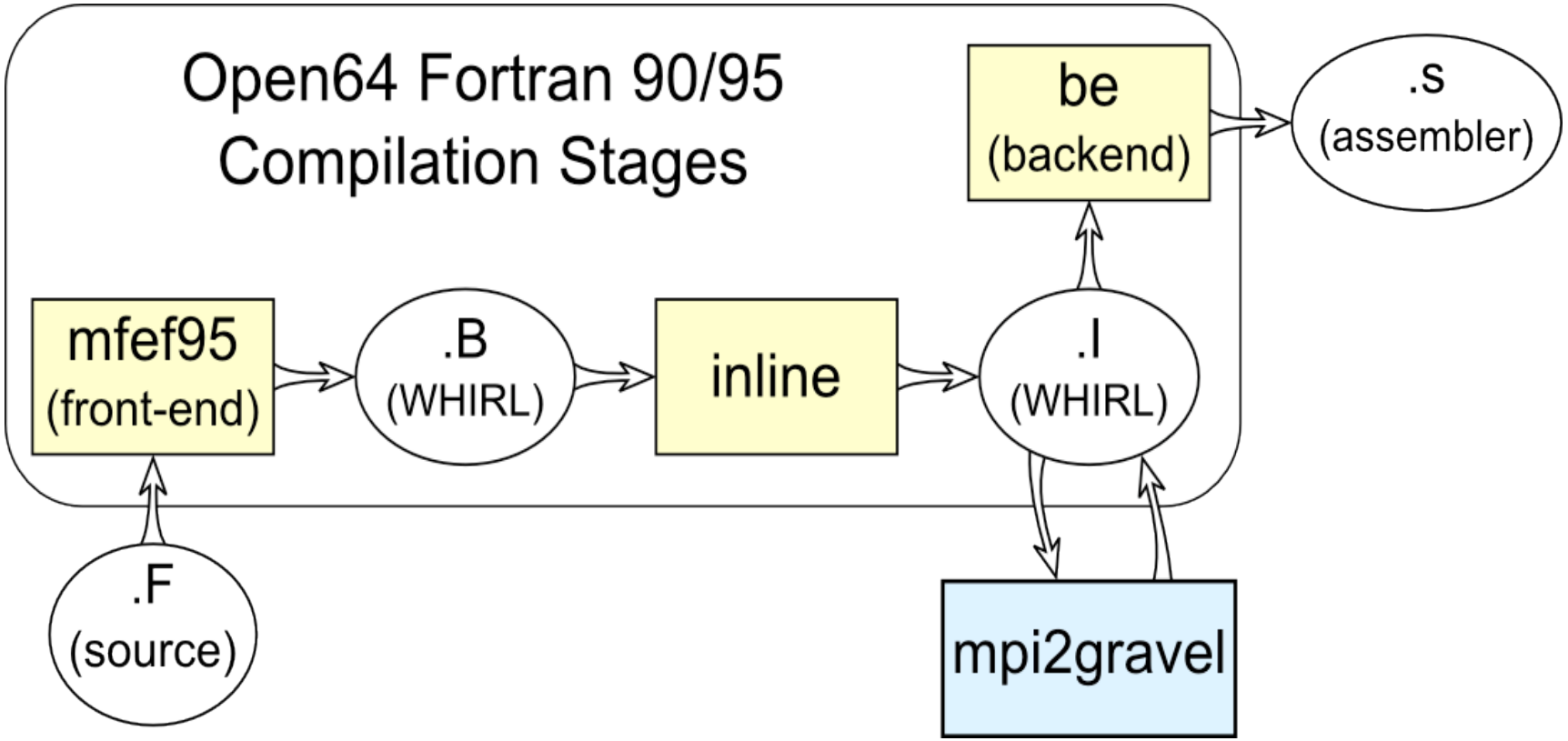


(c) Producer Initiated RDMA read protocol

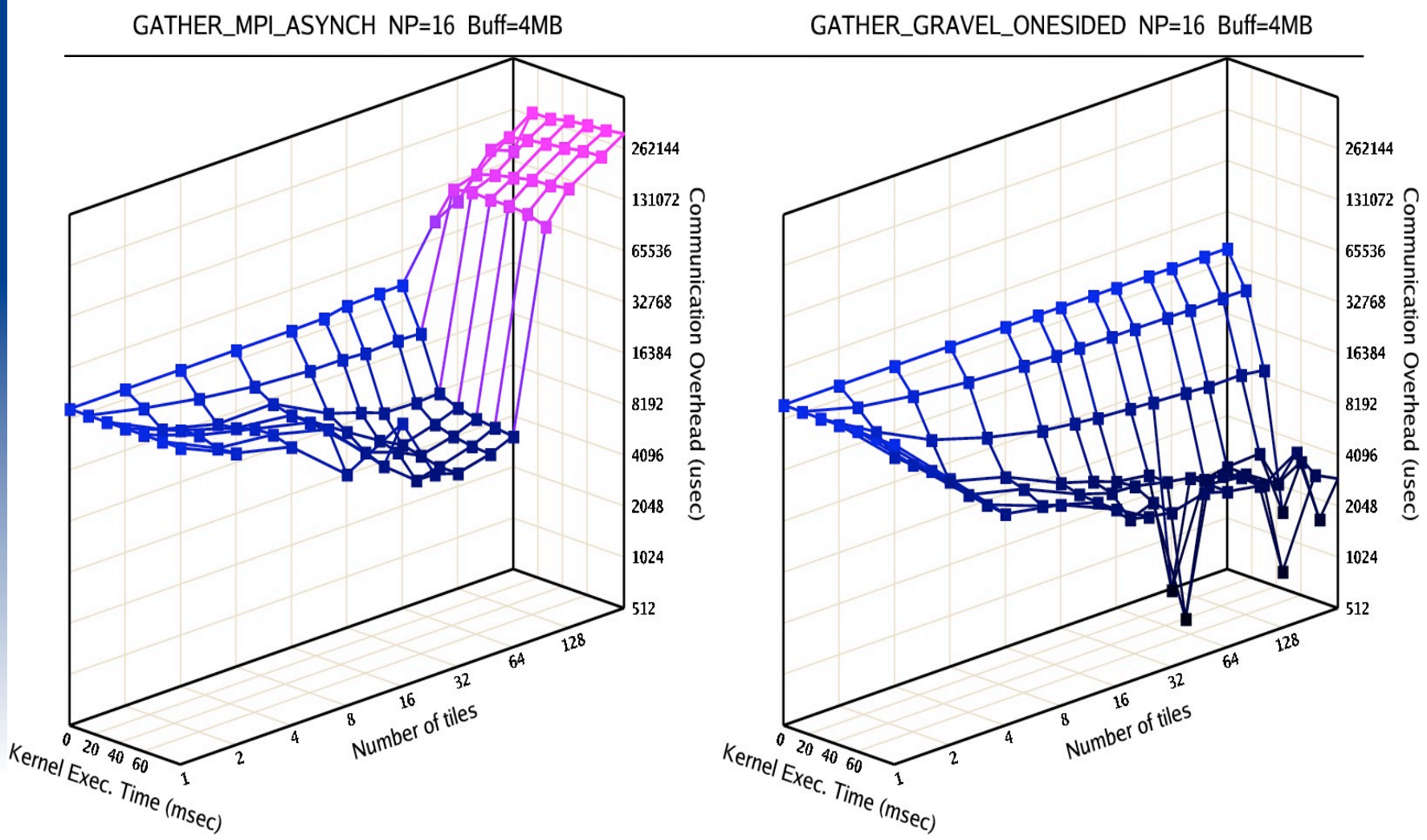


(d) Advanced RDMA write based protocol

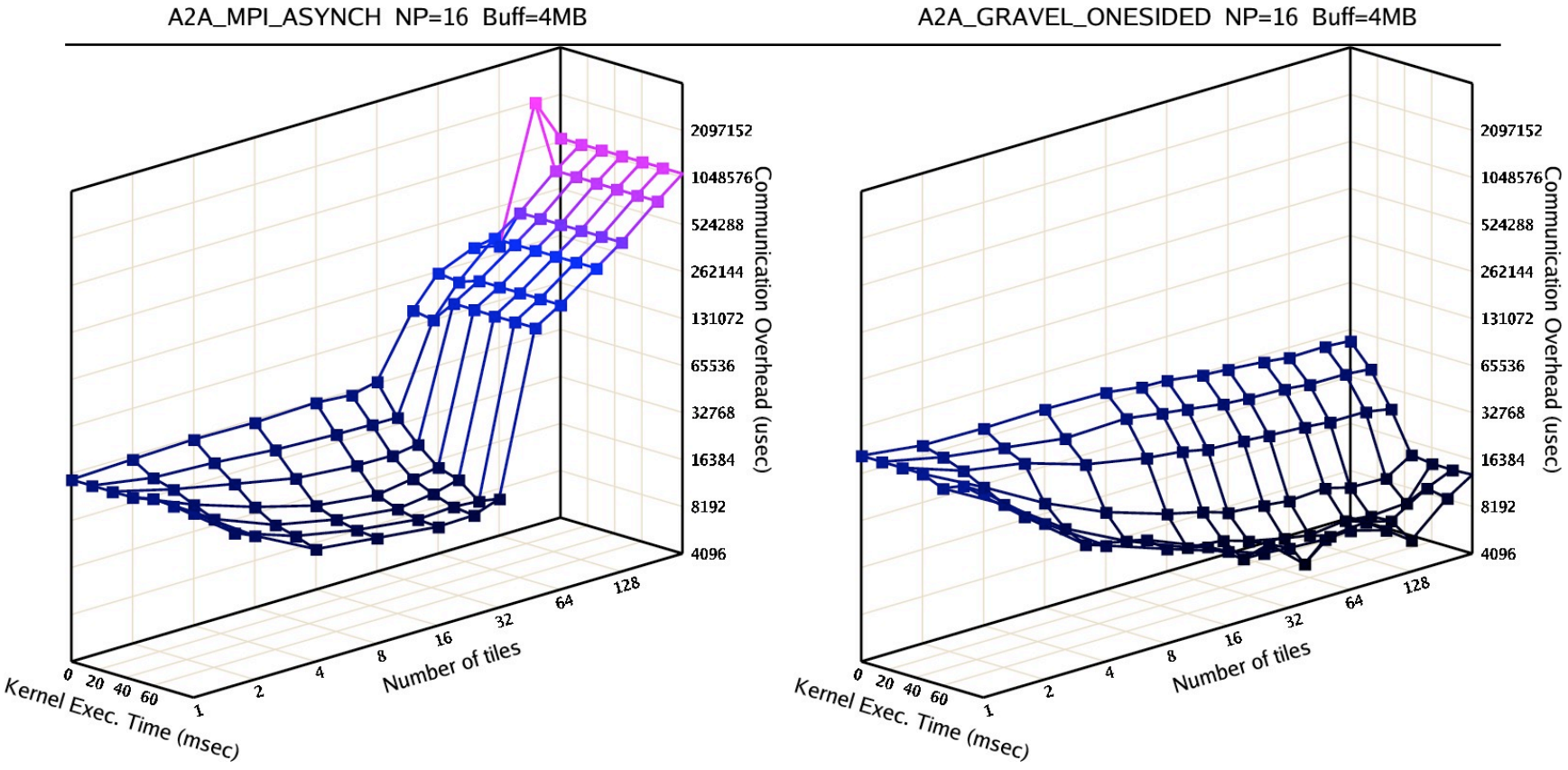
Open64 Implementation



Gravel Performance



Gravel Performance - 2



- Automatic Tuning of MPI Software
  - Martin Swany, Lori Pollock, U. Delaware
  - Jack Dongarra, George Bosilca, U. Tennessee
- For real codes, the MPI library must be aware that MPI calls have been removed
  - Only performance critical loops will likely be optimized
- Initial work: Optimized packing routines
- Next, make OpenMPI more “inline-friendly”

Autotuning

- We need to interleave computation and communication and that means co-tuning
- Models are difficult as the maximum bandwidth and minimum latency may not be the key factors when considering whole application network overhead -- runtime is the final metric!
  - We've mentioned reduction in overall communication, but that's not the only possible solution either
- One of the key arguments for autotuning in this space is that there are many factors in this space and analytical models are intractable
  - In addition, when this tuning is combined with compute library tuning, it gets worse
- Considering pipelined message-passing is key to performance improvement
  - We don't want separately optimized phases
  - The need to interleave and compose has been mentioned repeatedly

Compiler Support

- We're using Open64
  - Some have talked about everyone rolling their own transformation infrastructure
- The interaction with the system for loop transformation suggests that a tight integration is necessary
  - Subsequent phases shouldn't undo what we've done
- Source to source is good for portability but might leave opportunities on the table
  - We'd like to expand the ledger notion and potentially eliminate messaging call sites altogether

Acknowledgements

- UD Students
  - Anthony Danalis, Andrew Gearhart, Aaron Brown, Magnus Johnsson, Ben Perry, Omer Arap
    - (alumni: Lewis Fishgold, Kiyong Kim)
- co-PI: Lori Pollock
- NSF CSR Program
  - CNS-0509170
  - CNS-0720712