Dealing with the Scale Challenge

Presented by

George Bosilca

Innovative Computing Laboratory
University of Tennessee

SC11 Seattle, WA

ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering
and Computer Science

The Quest for Alternative Programming Paradigms

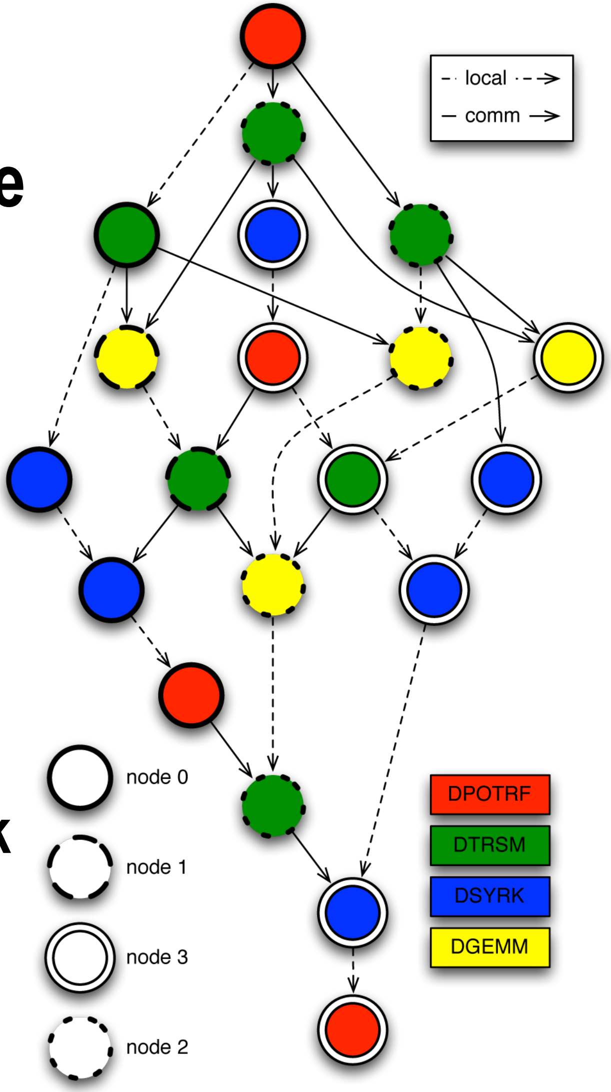
DAGuE / DPLASMA

Direct Acyclic Graph Unified Environment
Performance Portability across large-scale hybrid platforms

Algorithm described as task dependencies

- Algebraic, problem-size independent representation of the algorithms
- Data distribution is independent of the algorithm description

The runtime manage the data dependencies, task scheduling and data movement between nodes



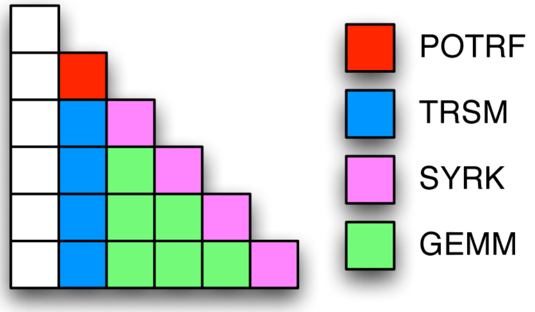
DAGuE/DPLASMA: Cholesky

```
FOR k = 0..TILES-1
    A[k][k] ← DPOTRF(A[k][k])
    FOR m = k+1..TILES-1
        A[m][k] ← DTRSM(A[k][k], A[m][k])
    FOR n = k+1..TILES-1
        A[n][n] ← DSYRK(A[n][k], A[n][n])
        FOR m = n+1..TILES-1
            A[m][n] ← DGEMM(A[m][k], A[n][k], A[m][n])
```

Original pseudo-code is converted by a preprocessor into DAGuE internal representation (shown below)

The DAGuE framework schedules the tasks based on the data flow dependencies, taking into account the architectural features of the underlying hardware (core and NUMA)

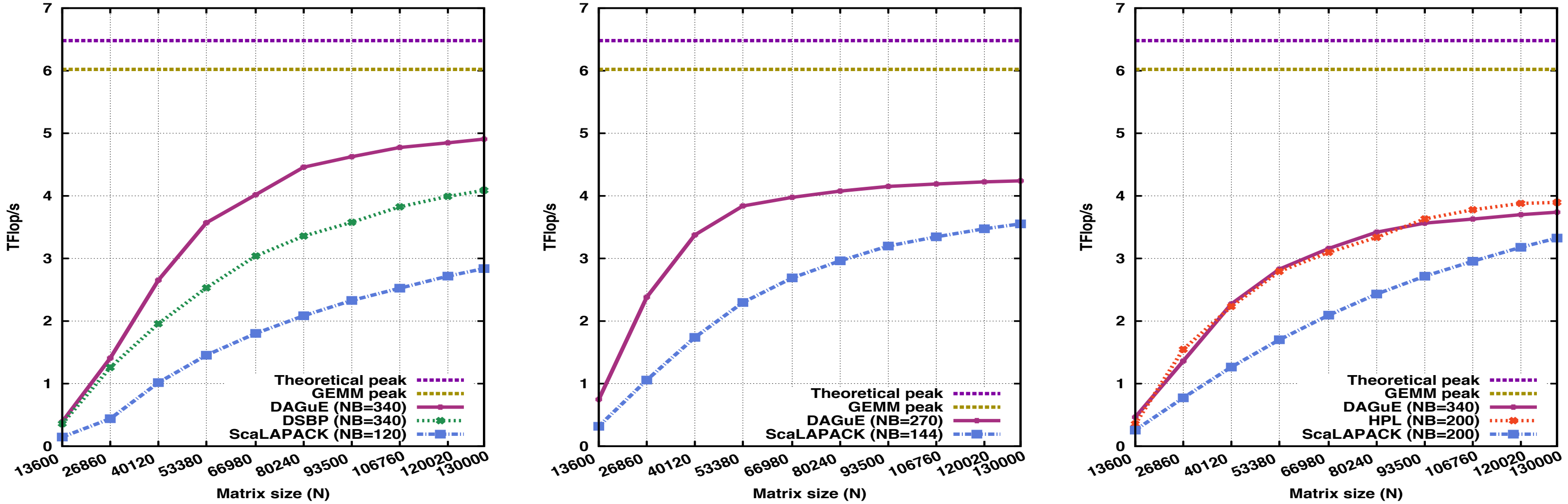
```
DPOTRF(k)  (high_priority)
  // Execution space
  k = 0..SIZE-1
  // Parallel partitioning
  : (k / rtileSIZE) % GRIDrows == rowRANK
  : (k / ctileSIZE) % GRIDcols == colRANK
  T <- (k == 0) ? A(k, k) : T DSYRK(k-1, k)    [TILE]
    -> T DTRSM(k, k+1..SIZE-1)                 [TILE]
    -> A(k, k)
```



Step *k* of Cholesky factorization

Scalability and Performance

One-sided factorizations on Griffon (81 nodes with 8 cores each)

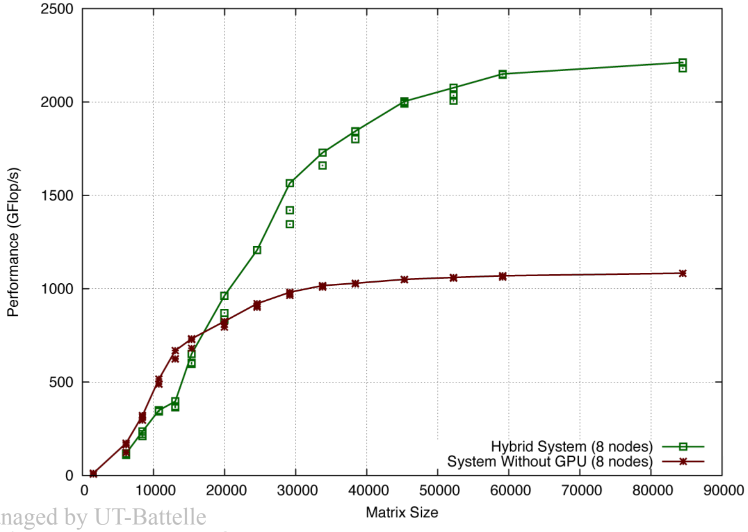


(a) Cholesky factorization.

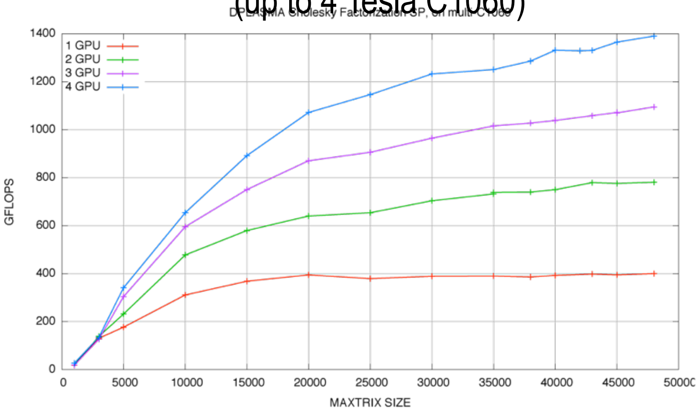
(b) QR factorization.

(c) LU factorization.

Cholesky on a GPU cluster (distributed 4 C2050, 4 C1060)



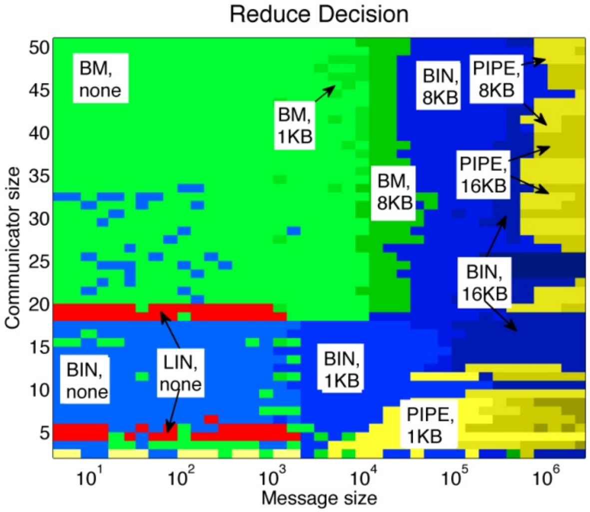
Cholesky on a single node multi-GPU (up to 4 Tesla C1060)



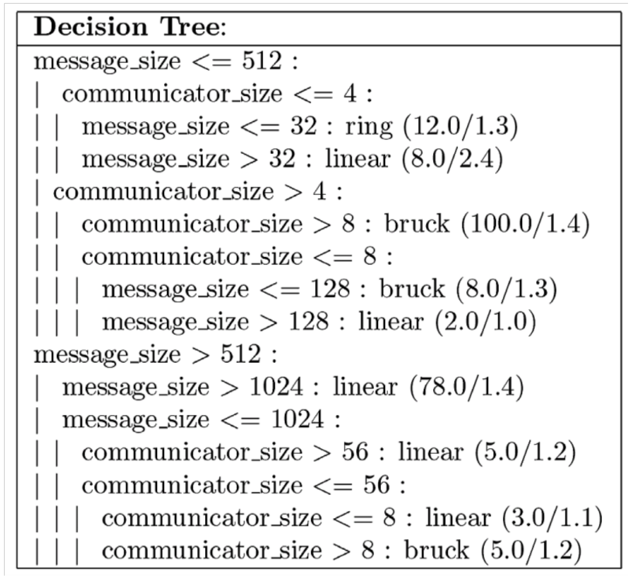
Optimized MPI Collective Communications

Optimization process

- **Minimize the collective communication execution time, by selecting the right algorithm based on the network characteristics and collective parameters (data size, number of processes)**
  - **We use performance models, graphical encoding, and statistical learning techniques to build platform-specific, efficient, and fast run-time decision functions**



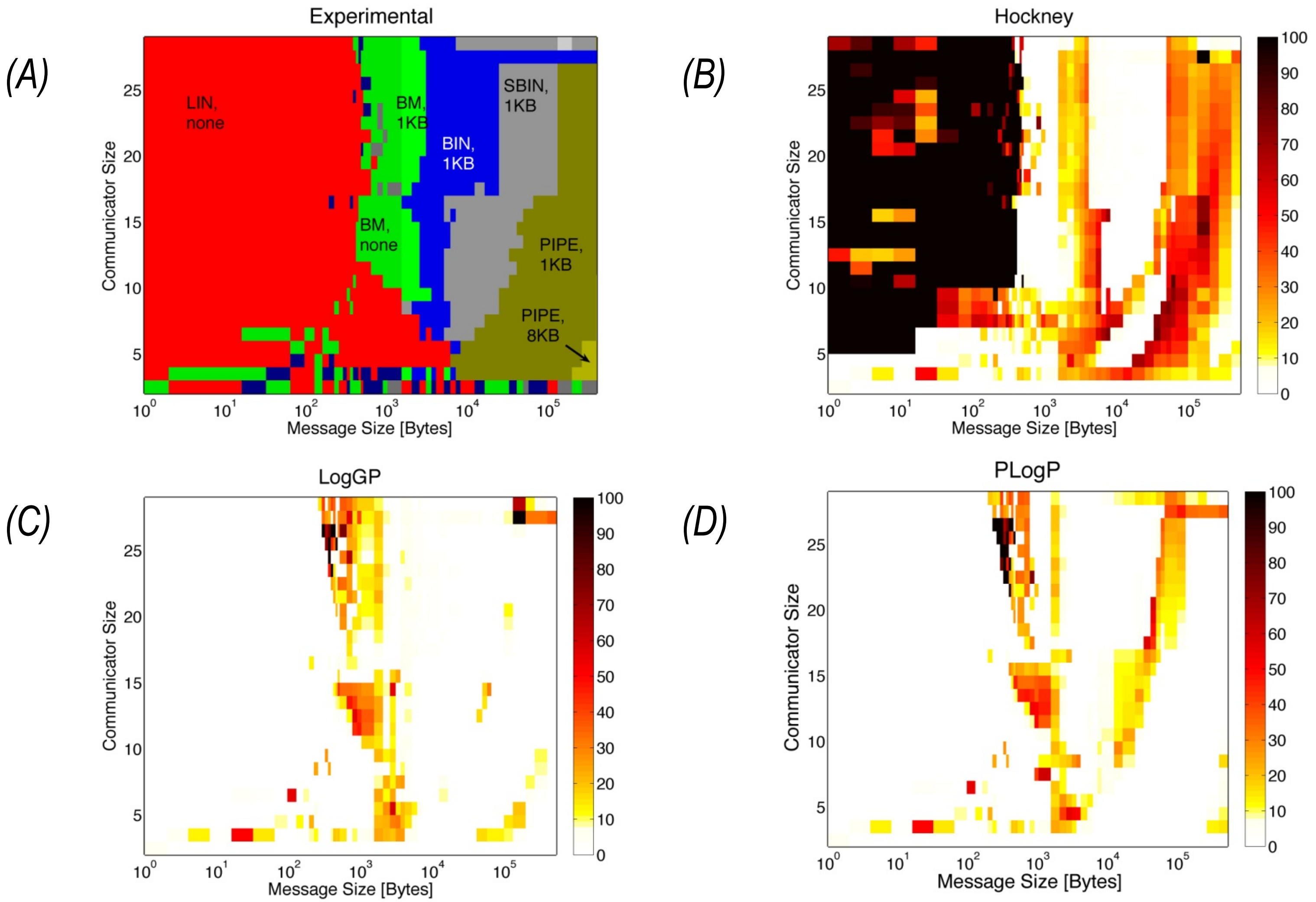
Fastest collective communications algorithms for a specific network depending on the message and communicator size



Decision Tree:

```
message_size <= 512 :
| communicator_size <= 4 :
| | message_size <= 32 : ring (12.0/1.3)
| | message_size > 32 : linear (8.0/2.4)
| communicator_size > 4 :
| | communicator_size > 8 : bruck (100.0/1.4)
| | communicator_size <= 8 :
| | | message_size <= 128 : bruck (8.0/1.3)
| | | message_size > 128 : linear (2.0/1.0)
message_size > 512 :
| message_size > 1024 : linear (78.0/1.4)
| message_size <= 1024 :
| | communicator_size > 56 : linear (5.0/1.2)
| | communicator_size <= 56 :
| | | communicator_size <= 8 : linear (3.0/1.1)
| | | communicator_size > 8 : bruck (5.0/1.2)
```

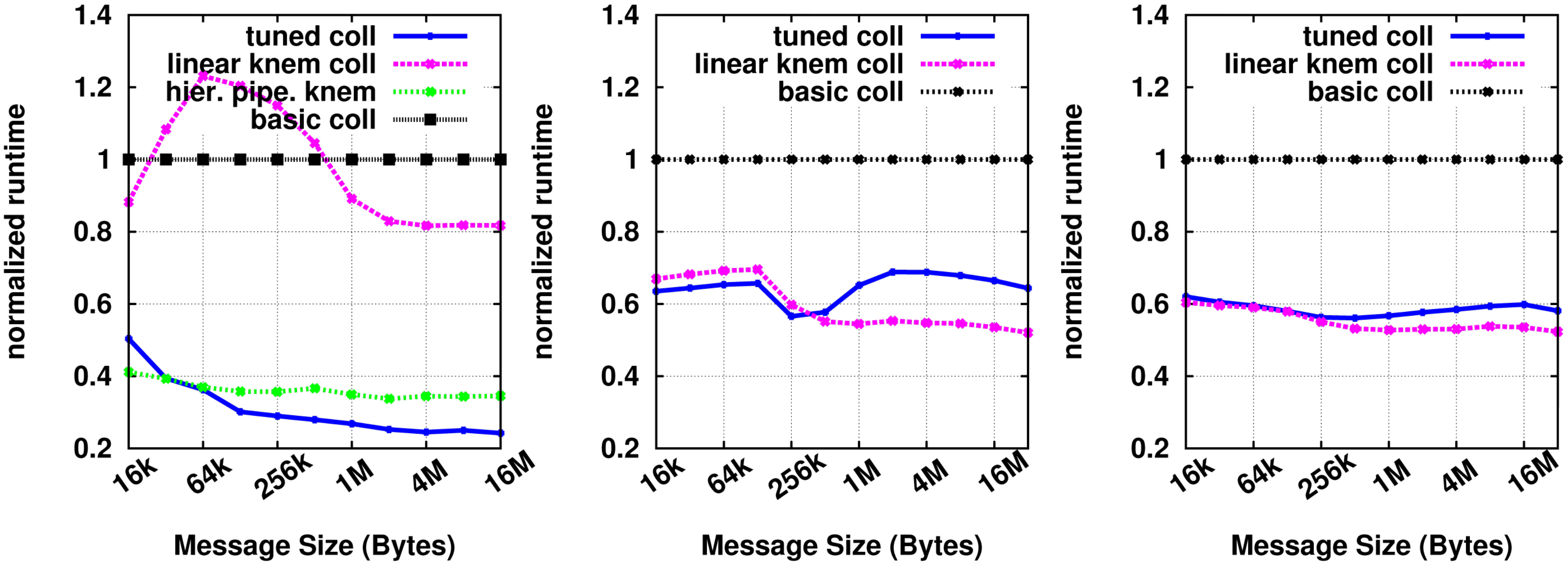
Model prediction vs. experimentation



Fastest collective communications algorithms for a specific network depending on the message and communicator size

Intra-node shared memory collectives

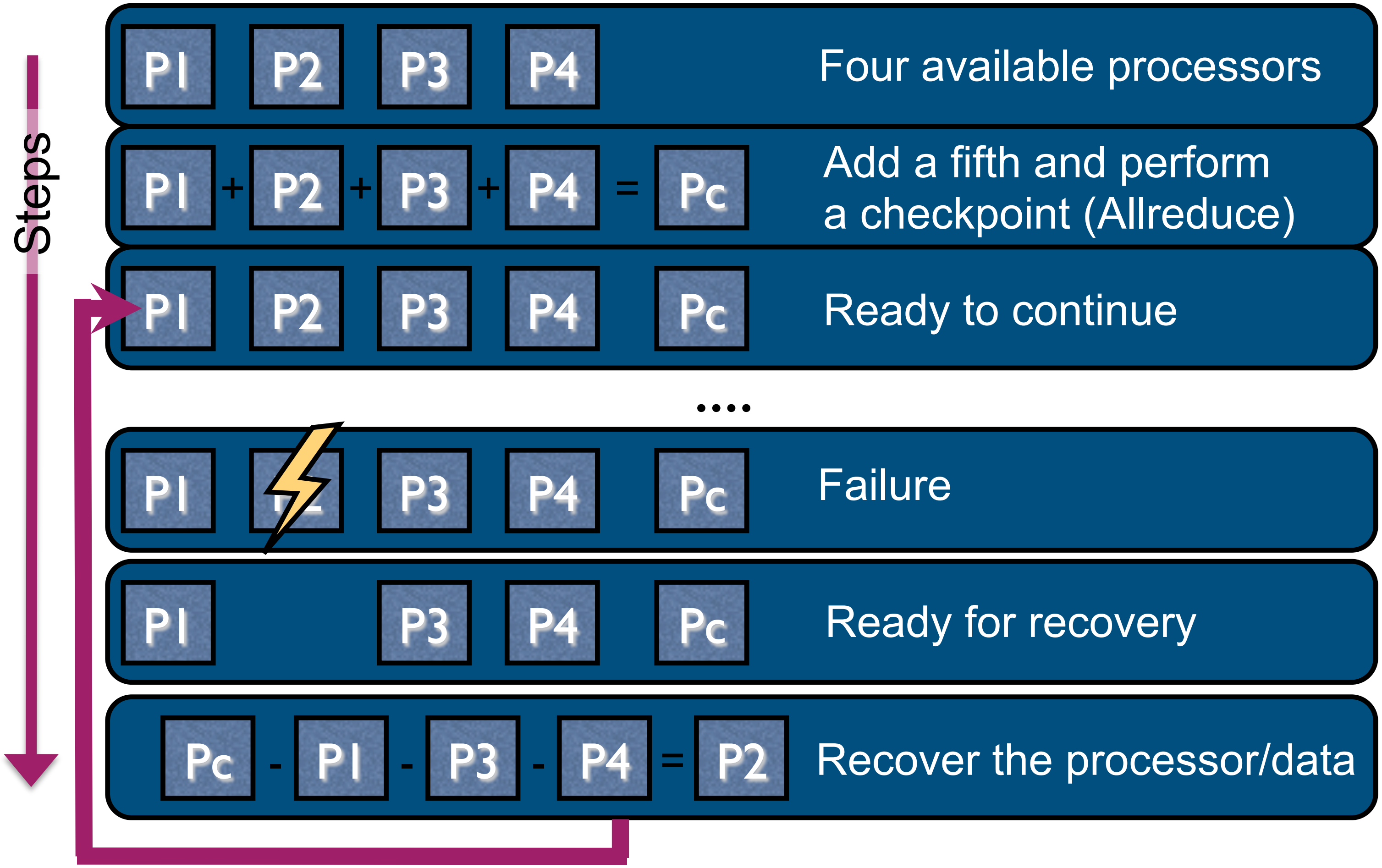
Memory node aware Allgather (normalized to default collective implementation)



- Taking advantage of the architecture features (cores and memory node placement) significantly improves collective communication performance
- Using knem for minimizing the number of memory copies
- HWLOC for accessing the information about the hardware capabilities

Fault Tolerance Diskless Checkpointing

Diskless checkpointing



Four available processors

Add a fifth and perform
a checkpoint (Allreduce)

Ready to continue

Failure

Ready for recovery

Recover the processor/data

Fault tolerance

Diskless checkpointing

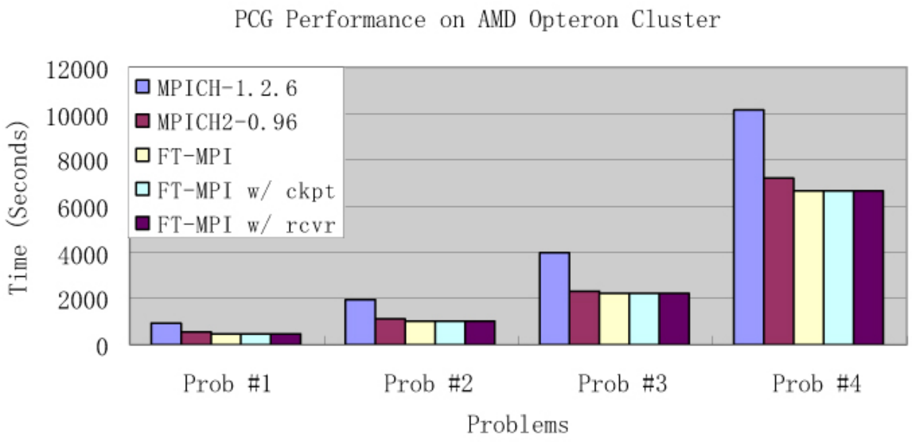
- How to checkpoint
  - Either floating-point arithmetic or binary arithmetic will work
  - If checkpoints are performed in floating-point arithmetic, then we can exploit the linearity of the mathematical relations on the object to maintain the checksums
- How to support multiple failures
  - Reed-Salomon algorithm
  - Support of *p* failures requires *p* additional processors (resources)

Fault Tolerant PCG

- **64×2 AMD 64 connected using GigE**

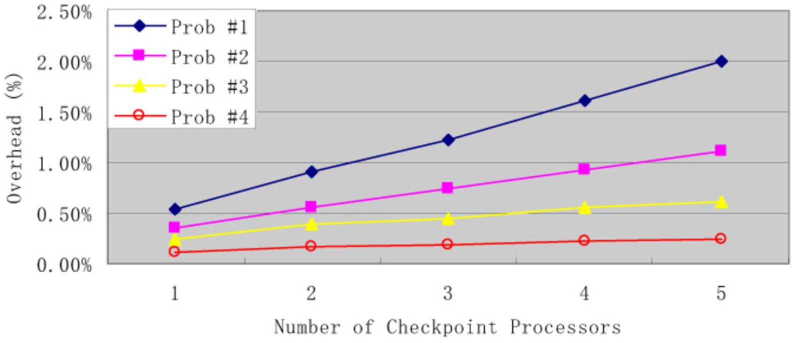
	Size of the Problem	Num. of Comp. Procs
Prob #1	164,610	15
Prob #2	329,220	30
Prob #3	658,440	60
Prob #4	1,316,880	120

Performance of PCG with different MPI libraries

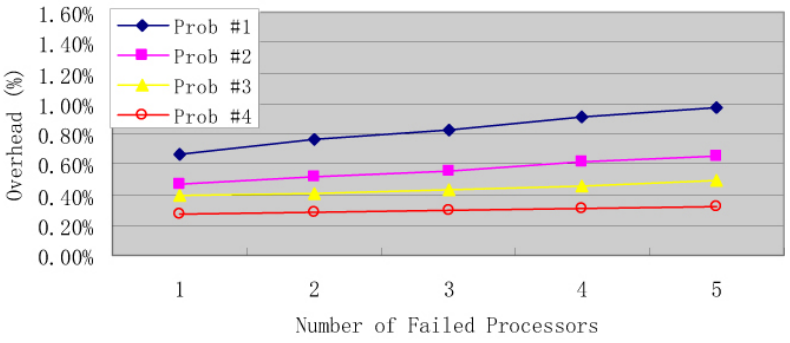


For checkpoint we generate one checkpoint every 2000 iterations

PCG Checkpoint Overhead

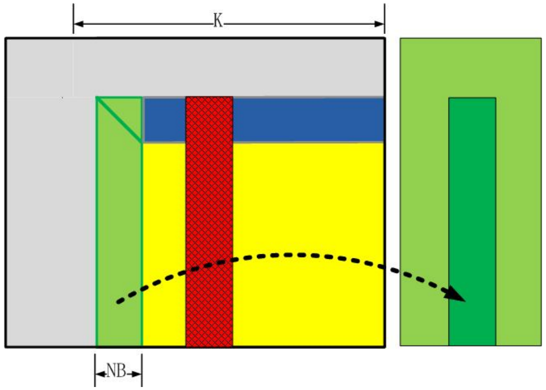


PCG Recovery Overhead

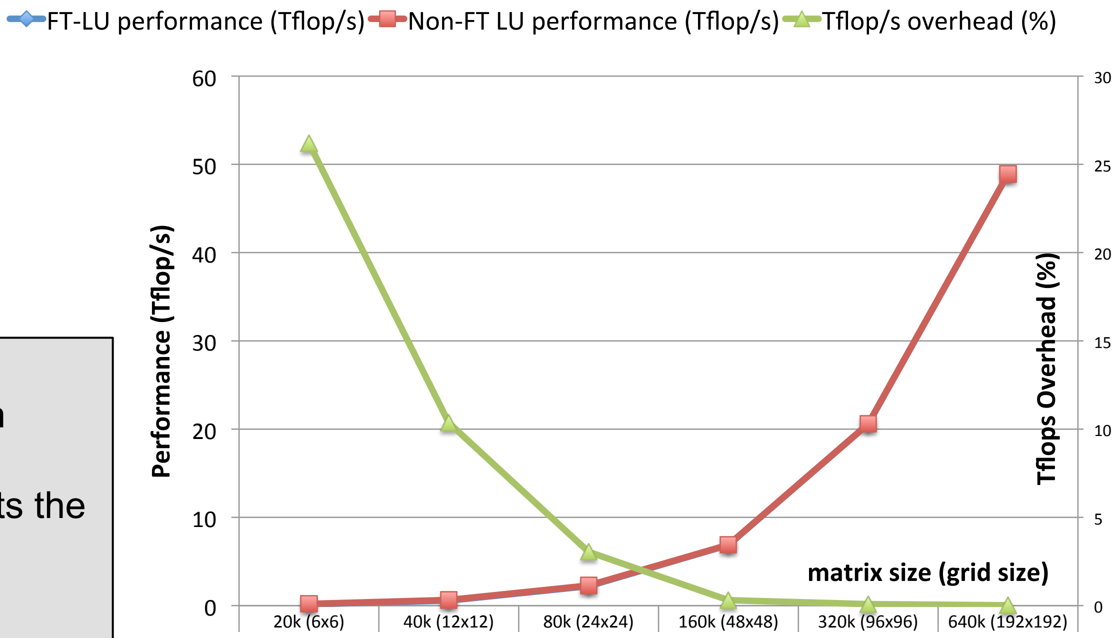


Algorithm Based Fault Tolerance - LU

$$\begin{bmatrix} A_{00} & A_{01} & \dots & A_{0N} & \sum_{k=1}^{N} A_{0k} \\ A_{10} & A_{11} & \dots & A_{1N} & \sum_{k=1}^{N} A_{1k} \\ \vdots & \vdots & & \vdots & \vdots \\ A_{M0} & A_{M1} & \dots & A_{MN} & \sum_{k=1}^{N} A_{Mk} \end{bmatrix} = \begin{bmatrix} Z_{00} & \dots \\ \vdots & \ddots \end{bmatrix} \begin{bmatrix} U_{00} & U_{01} & \dots & U_{0N} & \sum_{k=1}^{N} U_{0k} \\ U_{10} & U_{11} & \dots & U_{1N} & \sum_{k=1}^{N} U_{1k} \\ \vdots & \vdots & & \vdots & \vdots \\ U_{K0} & U_{K1} & \dots & U_{KN} & \sum_{k=1}^{N} U_{Kk} \end{bmatrix}$$



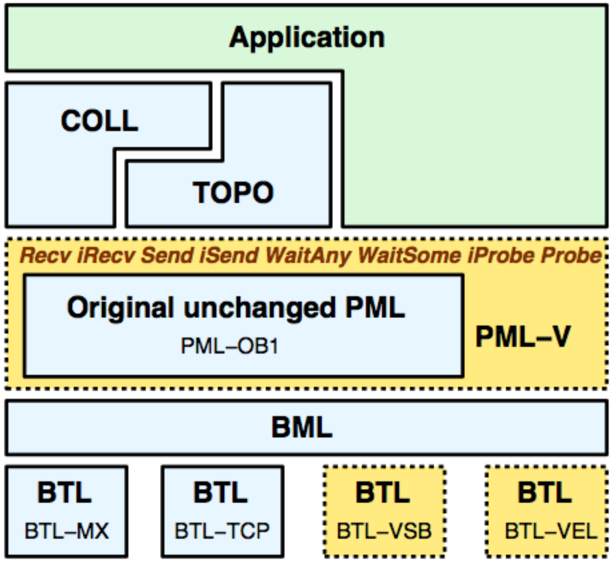
Gray: Result in previous steps
Light Green: Panel factorization result in current step
Deep Green: The checksum that protects the light green
Blue: TRSM zone
Yellow: GEMM zone
Red: one of the columns affected by pivoting



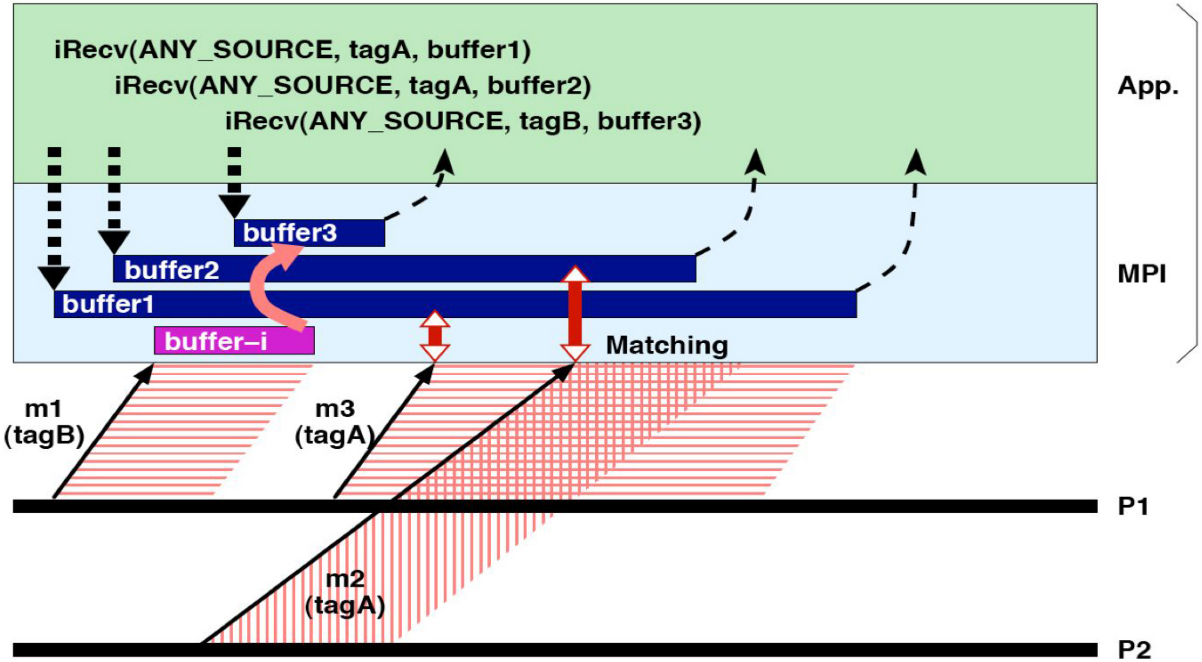
Automatic Fault Tolerance Using Message Logging

Interposition in Open MPI

- Vampire PML loads a new class of MCA components
  - Vprotocols provide the entire FT protocol (optimistic and pessimistic)
  - You can use the ability to define subframeworks in your components
- Keep using the optimized low level and zero-copy devices (BTL) for communication
- Unchanged message scheduling logic
- Generic framework where researchers can easily plug their own message logging–based fault tolerant approach



Detailing event types to avoid intrusiveness



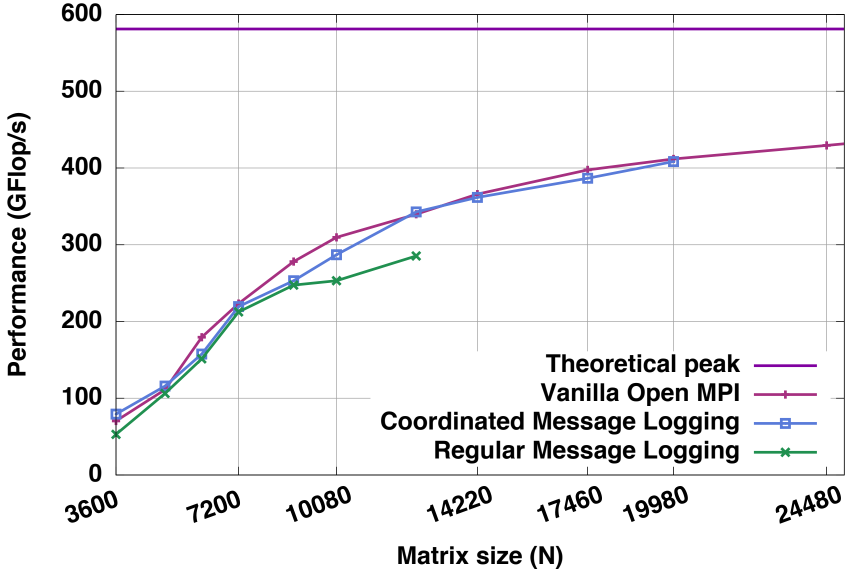
- Order of message receptions are non-deterministic events; messages received but not sent are inconsistent
- Possible loss of the whole execution and unpredictable fault cost
- Message logging enforces deterministic replay to restore a globally coherent state
- Protocol to avoid payload logging for correlated failure set (such as processes hosted on a shared memory environment)

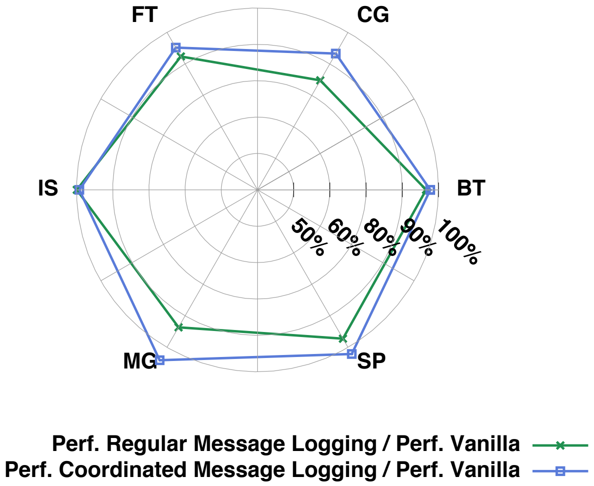
Performance overhead

	BT	SP	FT	CG	MG						LU					
#processors	all				4	32	64	256	512	1024	4	32	64	256	512	1024
%non-deterministic	0	0	0	0	40.33	29.35	27.10	22.23	20.67	19.99	1.13	0.66	0.80	0.80	0.75	0.57

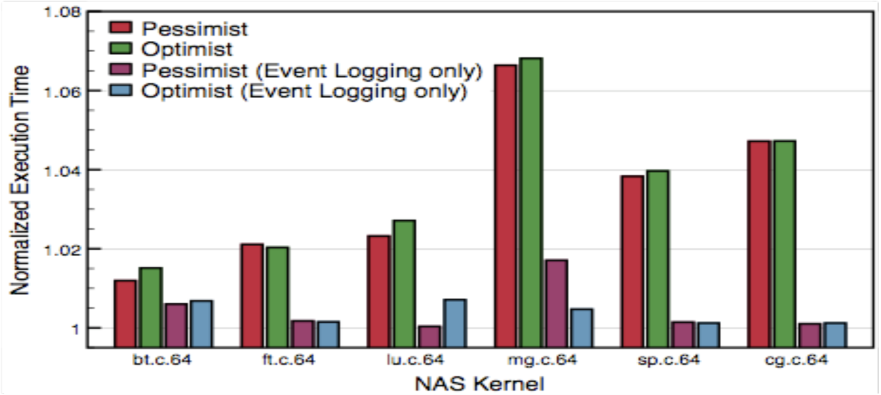
Table 1. Percentage of non-deterministic events to total number of exchanged messages on the NAS Parallel Benchmarks (class B)

HPL performance on multi-core clusters connected via Infiniband 20Gbs





NAS performance 32/36 cores over shared memory with correlated sets



Contact

George Bosilca

bosilca@eecs.utk.edu