Construction and Evaluation of Coordinated Performance Skeletons

(*Predicting Performance in an Unpredictable World***)**

Qiang Xu **Jaspal Subhlok**

University of Houston

HiPC 2008

Getting Started

OBJECTIVE: **Estimate application performance rapidly in a foreign/dynamic environment, e.g**

- Cluster with upgraded hardware or software components, e.g., MPI Library
- Desktop grid or “Volunteer nodes” or Amazon EC-2 cloud…
- Execution with different number of processes (8,16 or more processes best for 8 nodes)
- System under simulation

Common factor is a hard to model scenario

Skelton Based Approach ?

Build a short running "*skeleton*" program that mimics execution behavior of a given application

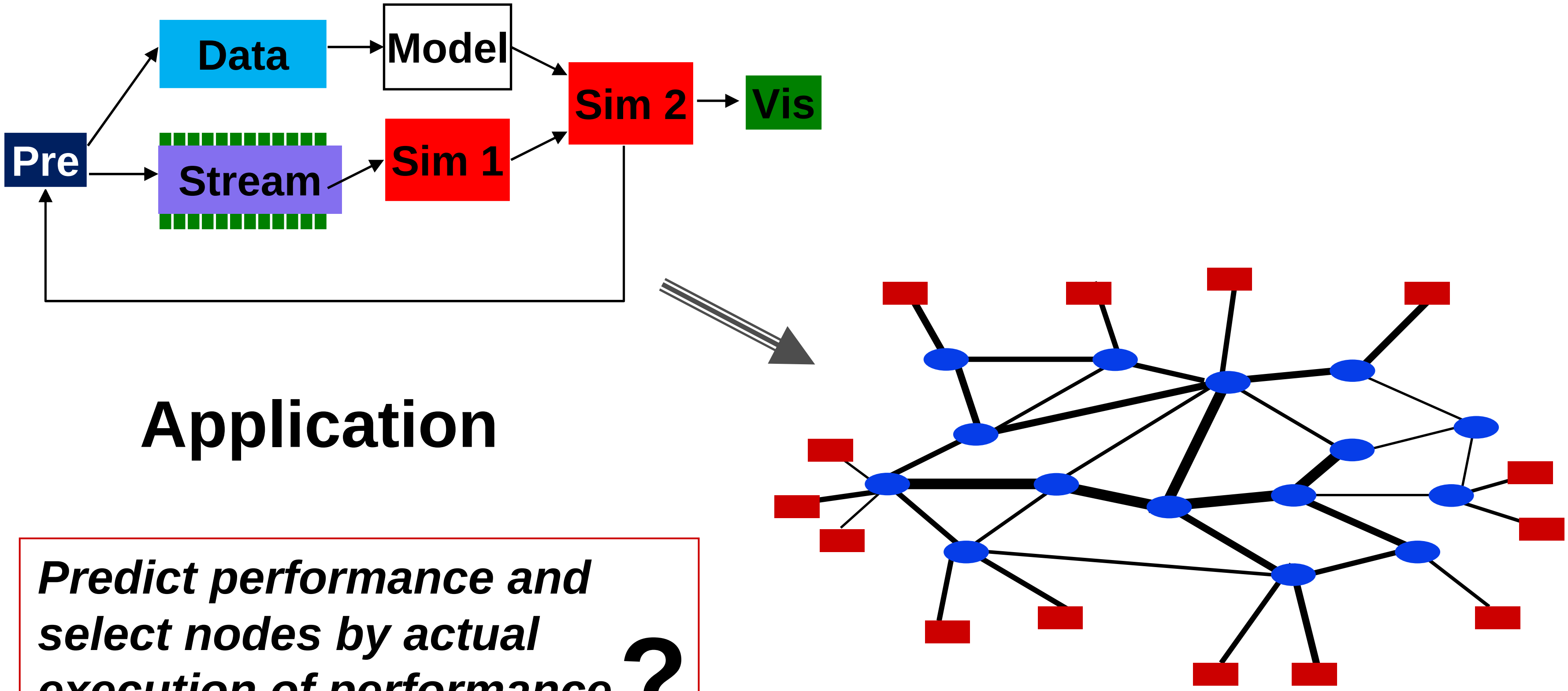
GOAL: execution time of a performance skeleton is a fixed fraction of application execution time - say 1:1000, then..

If the Application runtime is:	**Skeleton runs in:**
10K seconds on a dedicated compute cluster	10 secs
8K seconds with Open MPI on that cluster	8 secs
20K seconds on a shared heterogeneous grid	20 secs
1 million seconds under simulation	1000 secs
…..,	

Timed execution of a performance skeleton provides an estimate of application performance!

CS@UH

One Motivation: Mapping Distributed Applications on Networks

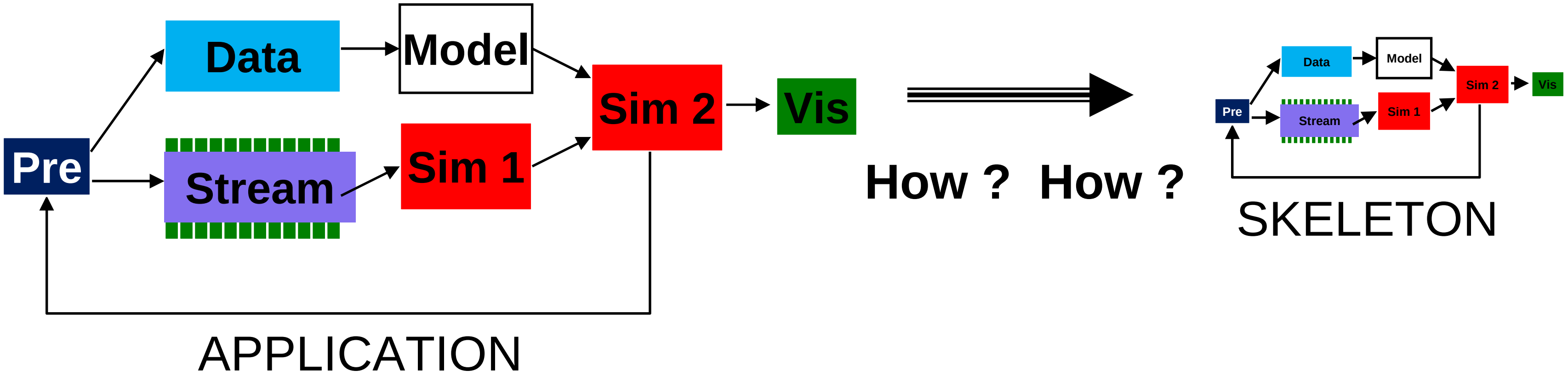


Application

Predict performance and select nodes by actual execution of performance skeletons on groups of nodes **?**

Network

How to Construct a Performance Skeleton ?



Pre
Data
Model
Stream
Sim 1
Sim 2
Vis

APPLICATION

How ? How ?

Pre
Data
Model
Stream
Sim 1
Sim 2
Vis

SKELETON

Central challenge in this research

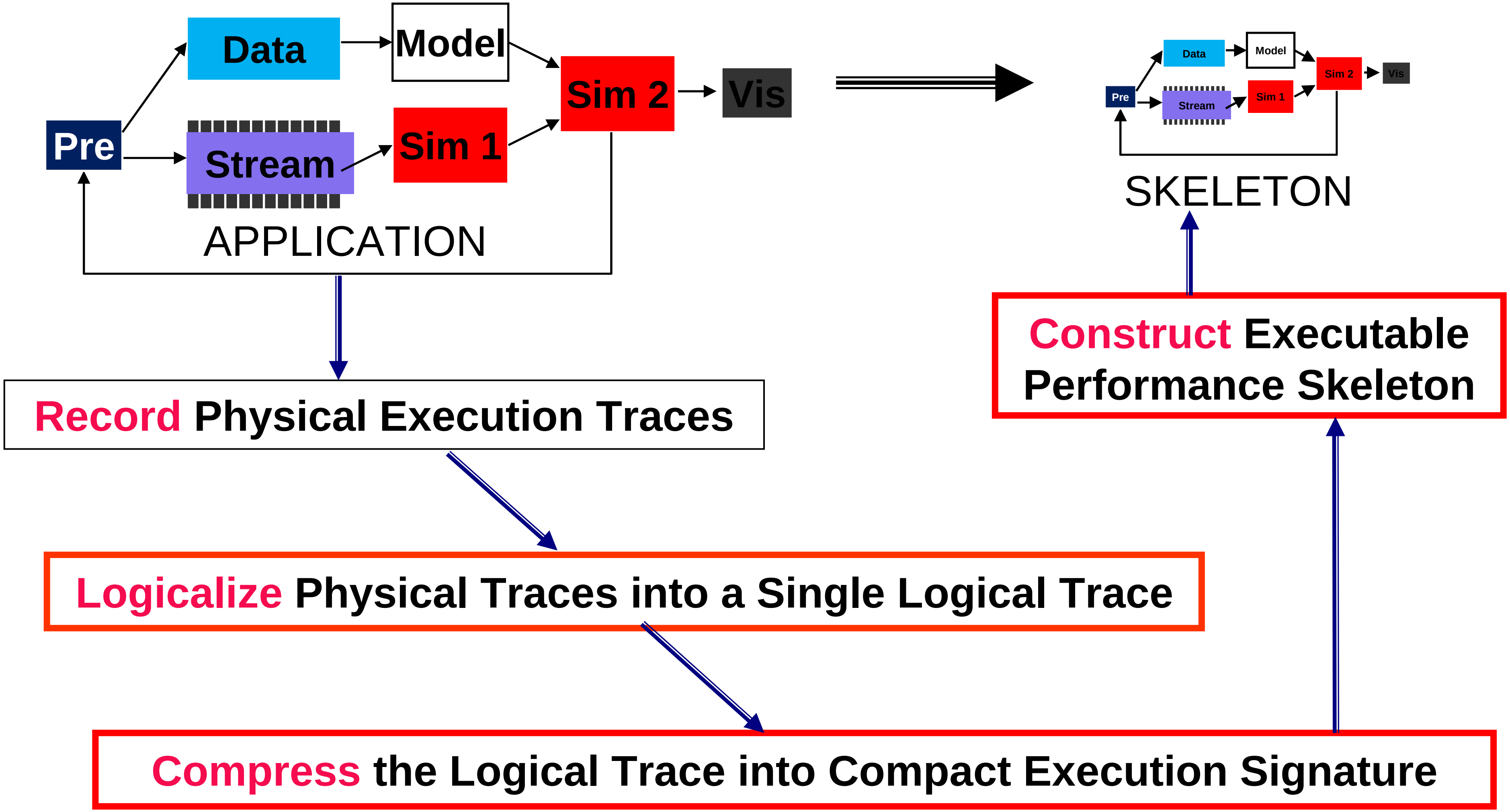
Common sense dictates that an application and its skeleton must be similar in:

- **Computation behavior**
- **Communication behavior**
- Memory behavior (partly addressed in related work)
- I/O Behavior (not directly addressed)

All execution behavior is to be captured in a **short program**

Skeleton Construction

Implementation for parallel MPI codes

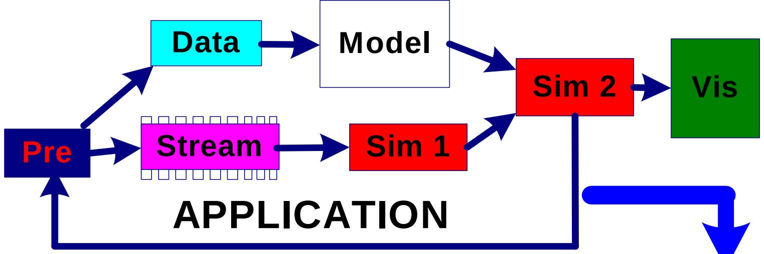


Record Physical Execution Traces

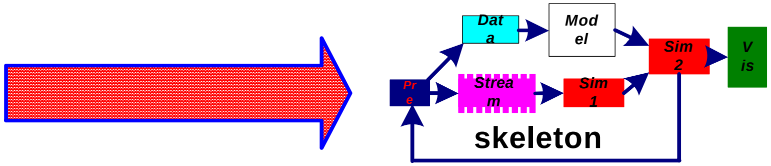
Logicalize Physical Traces into a Single Logical Trace

Compress the Logical Trace into Compact Execution Signature

Construct Executable Performance Skeleton



APPLICATION



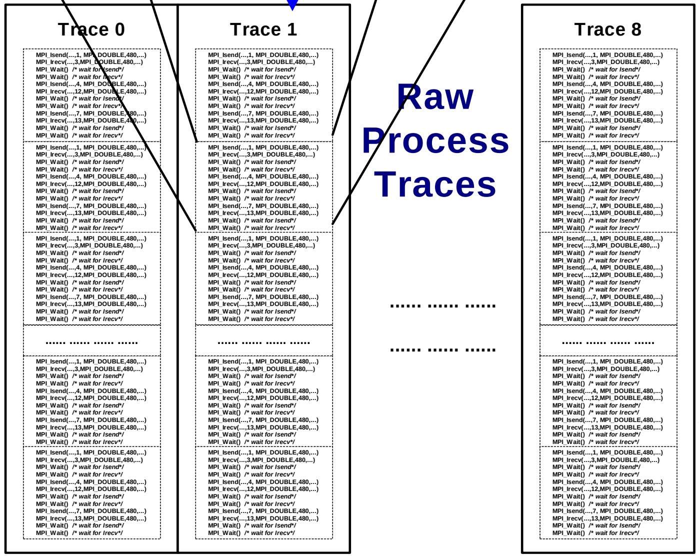
skeleton

$$\ldots\ \{AB[Z]^2CD[Z]^2EF[Z]^2\}^N \ldots \rightarrow \ldots\ \{AB[Z]^2CD[Z]^2EF[Z]^2\}^{n=N/100} \ldots$$

Loop Discovery

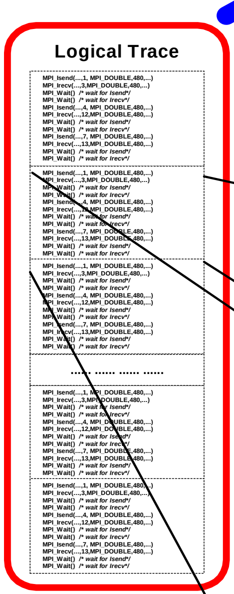
```
... ... ABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZ
CDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZ
ABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZ
EFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZ
CDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZ
ABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZ
EFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZ
CDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZABZZCDZZ ... ...
```

```
MPI_Isend(...,2, MPI_DOUBLE,480,...)
MPI_Irecv(...,0,MPI_DOUBLE,480,...)
MPI_Wait()  /* wait for Isend*/
MPI_Wait()  /* wait for Irecv*/
MPI_Isend(...,4, MPI_DOUBLE,480,...)
MPI_Irecv(...,7,MPI_DOUBLE,480,...)
MPI_Wait()  /* wait for Isend*/
MPI_Wait()  /* wait for Irecv*/
MPI_Isend(...,3, MPI_DOUBLE,480,...)
MPI_Irecv(...,8,MPI_DOUBLE,480,...)
MPI_Wait()  /* wait for Isend*/
MPI_Wait()  /* wait for Irecv*/
```

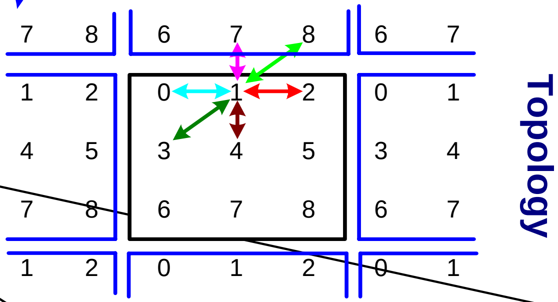


Raw Process Traces

Logicalization



Single Logical Trace



```
MPI_Isend(...,EAST, MPI_DOUBLE,480,...)       A
MPI_Irecv(...,WEST,MPI_DOUBLE,480,...)        B
MPI_Wait()  /* wait for Isend*/               Z
MPI_Wait()  /* wait for Irecv*/               Z
MPI_Isend(...,SOUTH, MPI_DOUBLE,480,...)      C
MPI_Irecv(...,NORTH,MPI_DOUBLE,480,...)       D
MPI_Wait()  /* wait for Isend*/               Z
MPI_Wait()  /* wait for Irecv*/               Z
MPI_Isend(...,SOUTHWEST, MPI_DOUBLE,480,...)  E
MPI_Irecv(...,NORTHEAST,MPI_DOUBLE,480,...)   F
MPI_Wait()  /* wait for Isend*/               Z
MPI_Wait()  /* wait for Irecv*/               Z
```

CS@UH

Logicalization

(N Physical Traces → Single Logical Trace)

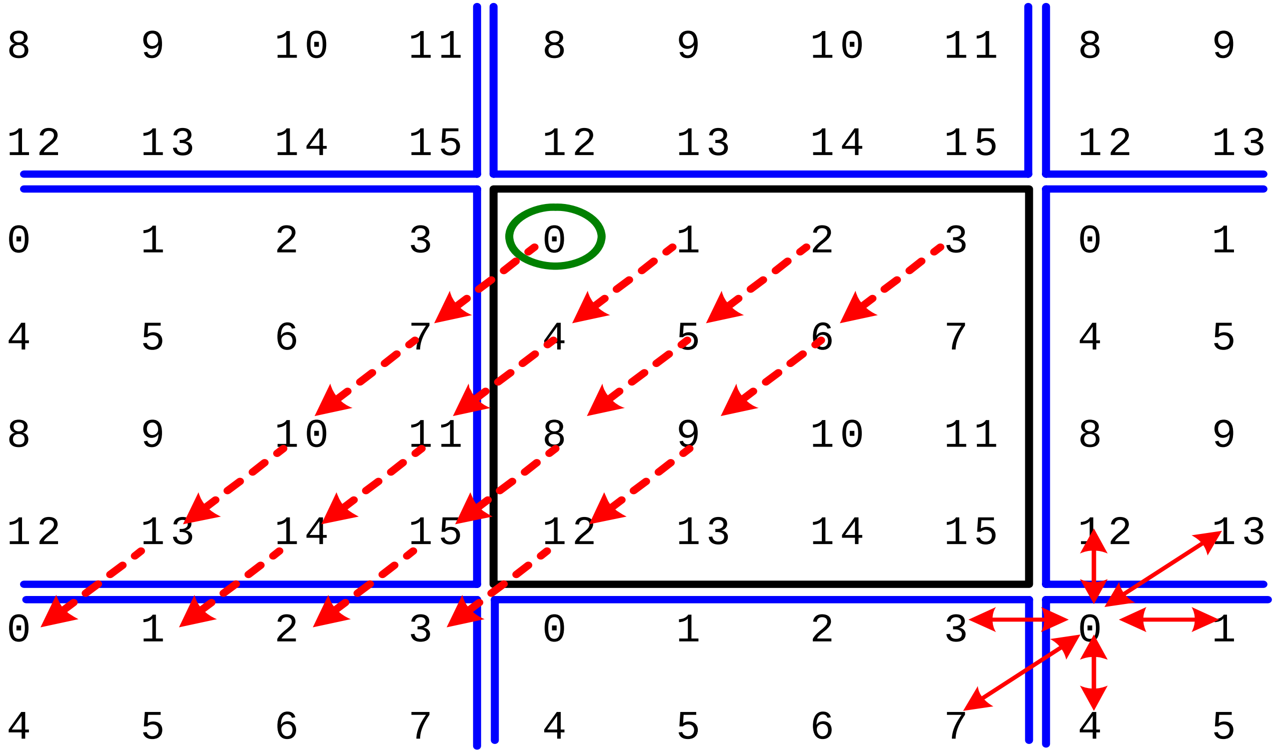
- **In SPMD point-to-point communication all processors typically perform the same communication….**

  … on different data and with different processors (e.g. Left/Right neighbors)

  A regular logical communication topology exists (e.g. *Grid, Torus, Stencil, Hypercube, Butterfly*…)

Logicalization identifies the logical topology to convert family of physical traces to a logical trace,

An Example – 16-process BT benchmark



8	9	10	11	8	9	10	11	8	9
12	13	14	15	12	13	14	15	12	13
0	1	2	3	0	1	2	3	0	1
4	5	6	7	4	5	6	7	4	5
8	9	10	11	8	9	10	11	8	9
12	13	14	15	12	13	14	15	12	13
0	1	2	3	0	1	2	3	0	1
4	5	6	7	4	5	6	7	4	5

P0: Send (7, data), P1: Send (4, data) P2:: Send (5, data) ,…. P15: Send (2, data) **➔** ***Send (SW, data)***

in logical trace in the context of a 2D torus topology

Logicalization

Key challenge: Identify dominant communication topology from inter-node communication matrix

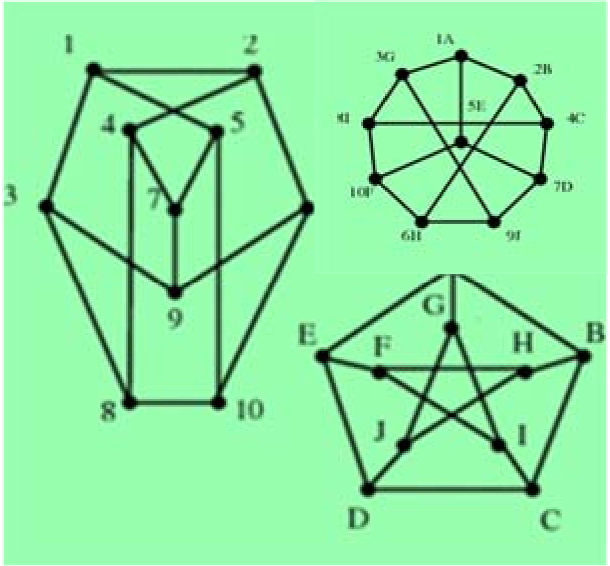
Matching against a known topology

Is solving graph isomorphism

- *No polynomial algorithm*

Practical solution with 3 Tests:

1. **Match node & edge counts**
2. **Match eigenvalues**
3. **Graph Isomorphism algorithm: employed VF2 library**

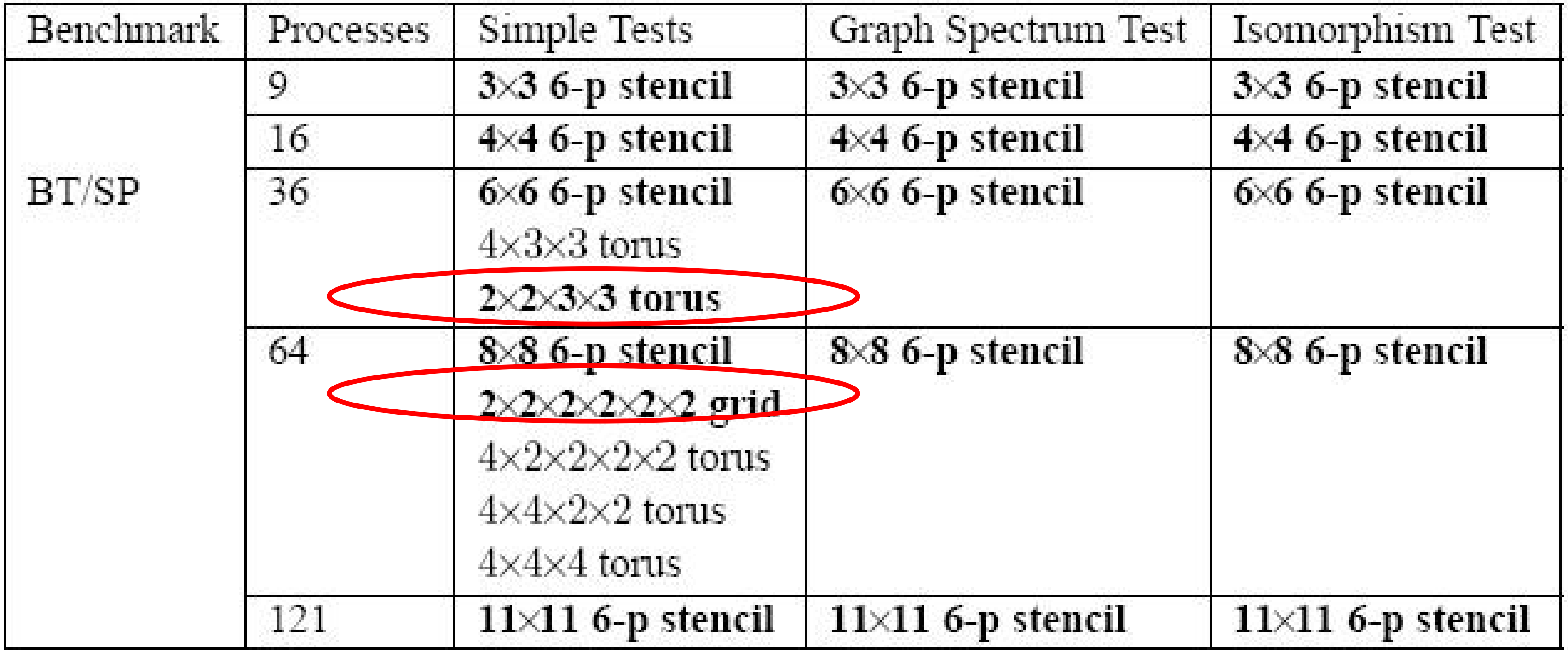


Test 1 eliminates most patterns cheaply.

Test 2 and Test 3 expensive but used sparingly.

Only Test 3 proves that a match exists.

Illustration: BT/SP Benchmark



Benchmark	Processes	Simple Tests	Graph Spectrum Test	Isomorphism Test
BT/SP	9	**3×3 6-p stencil**	**3×3 6-p stencil**	**3×3 6-p stencil**
	16	**4×4 6-p stencil**	**4×4 6-p stencil**	**4×4 6-p stencil**
	36	**6×6 6-p stencil** 4×3×3 torus **2×2×3×3 torus**	**6×6 6-p stencil**	**6×6 6-p stencil**
	64	**8×8 6-p stencil** **2×2×2×2×2×2 grid** 4×2×2×2×2 torus 4×4×2×2 torus 4×4×4 torus	**8×8 6-p stencil**	**8×8 6-p stencil**
	121	**11×11 6-p stencil**	**11×11 6-p stencil**	**11×11 6-p stencil**

- Table shows candidate topologies remaining after each test
- Non-boldface topologies are isomorphic to topology above

Logicalization Notes

Works well in practice!

- Main communication topology must be static & regular
- Matching only against known patterns, but patterns easy to add and library can be large
    - All n-dim grids or n-ary trees specified in one shot
- Some message exchange not related to main communication pattern observed
    - Ignored with thresholding, only dominant toplogies captured
- Multiple mixed patterns (equal to subgraph isomorphism) not yet implemented

More details: *Q. Xu, R. Prithivathi, J. Subhlok, and R.Zheng, Logicalization of MPI communication traces, TRUH-CS-08-07, Univ of Houston, May 08*

Compression of Logical Trace

Goal is to identify loop nests in the trace!

Matching sliding windows of trace is O(N^3).
-- Commonly employed locally on trace sections
-- So can miss long range repeats (outer loops).

Two new algorithms developed:

1. An optimal O(N^2) algorithm (finds outer loops first) : leverages Crochemore's algorithm to find all repeats

2. Greedy algorithm (finds inner loops first) guaranteed to miss at most 2 iterations of a loop – Very fast

Loop Discovery Performance

NAS Class C	Raw Trace Length (MPI Calls)	Compressed Trace Length (MPI Calls)	Optimal Loop Discovery (seconds)	Greedy Loop Discovery (seconds)
BT	17106	44	311.18	8.91
SP	26888	89	747.73	7.61
LU	323048	63	***113890.21*** ***(~30 hours)***	***61.9***
CG	41954	10	240.27	8.48
MG	10047	648	144.54	10.88

More details: *Q. Xu and J. Subhlok., Efficient discovery of loop nests in communication traces of parallel programs, TR UH-CS-08-08, Univ of Houston, May 2008*

Skeleton Code Generation

Compressed logicalized trace, i.e., loop nest of MPI calls and compute operations

TO

Compact Matching executable C code

1. MPI calls in trace converted to executable MPI calls on synthetic data – global SPMD communication pattern generated

2. Compute sections converted to synthetic computations of equal duration

3. # of iterations in loop nest reduced to match desired skeleton execution time

Code Generation Challenges

- **“Local” communication**
  - e.g., No matching Send in trace for a Recv

The send is ignored or a synthetic matching Recv is generated

- **“Unbalanced” communication**
  - Send and Recv not matching, e.g., in data size

Match forced by adjusting parameters, e.g using mean data size.

These represent exceptions to the global dominant communication pattern.

Code generator ensures correctness with possible inaccuracy

Validation of Skeleton Construction

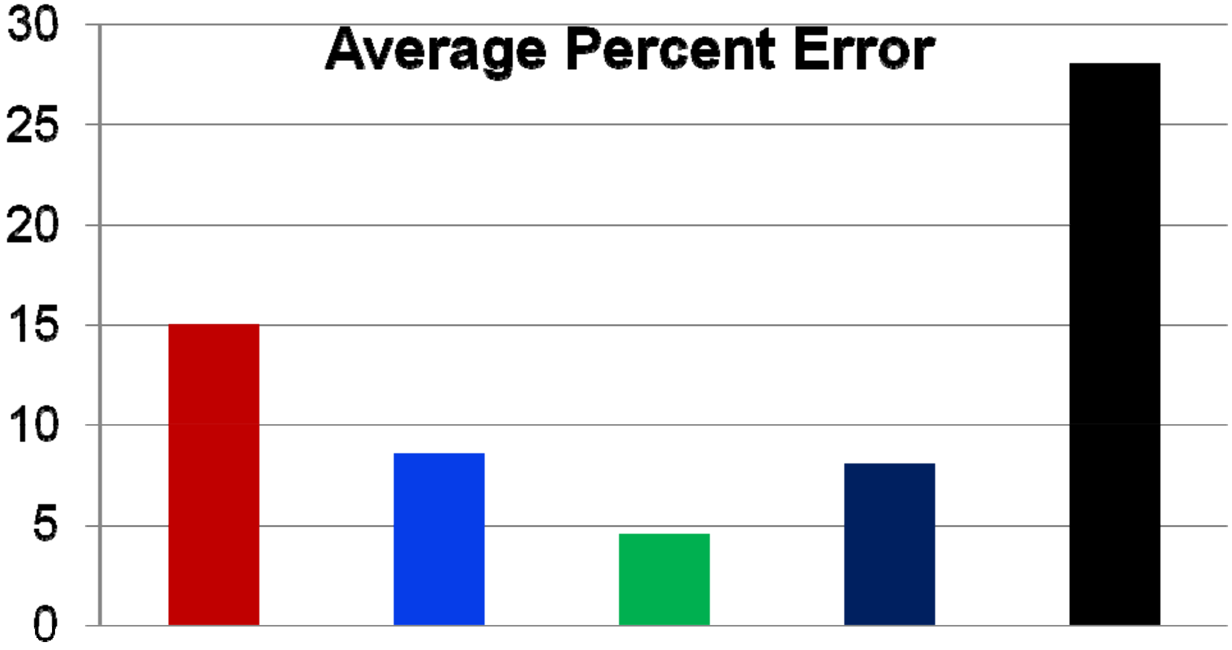
Skeletons constructed for Class C NAS MPI benchmarks up to 128 nodes

Skeletons constructed in one scenario → Employed to predict performance in a new scenario:

- **Execution on a different cluster**
- **Execution under a new MPI library**
- **Execution under varying available bandwidth**
- **Execution with different number of nodes for the same number of processes**
- **Execution under competition with other jobs**

Validation Results

Summary from a large suite of experiments!



Across Cluster Archs (1.7 GHz Xeon --> 2.3 GHZ Dual Core Opteron)

Across Communication Libraries (MPICH --> Open MPI)

Simulate Bandwidth Sharing (100 Mbps --> 5, 20, 50, Mbps)

Processor Sharing within Application (1--2, 4 processes/processor)

Processor Sharing with External Apps (add1, 2 competing processes)

Validation Results

For most applications and scenarios, the prediction was rather accurate with error within 10% for skeletons running for a few minutes

However:

- Prediction in some scenarios is inaccurate

Reasons:

1. Computing not modeled precisely (memory, instructions)
2. Synchronization impact can exaggerate variations

Conclusions

p

- Performance skeletons are an effective tool for estimating performance where modeling is impractical
- Methodologies for logicalization and loop nest discovery have broad applicability

FOR MORE INFORMATION (including papers/TRs with details of logicalization and compression):

- **www.cs.uh.edu/~jaspal** **jaspal@uh.edu**

Thanks to NSF

CS@UH