Exploiting Epochs and Symmetries in Analysing MPI Programs

Rishabh Ranjan, Ishita Agrawal, Subodh Sharma
IIT Delhi, India

Automated Software Engineering, 12 Oct 2022, Rochester, MI, USA

Introduction

Motivation

Verify message-passing (MP) programs for deadlocks

- MP programs are prevalent in high-performance scientific computing, etc.
- The curse of deadlocks:
  easy to introduce, hard to detect, catastrophic manifestations
- Non-determinism makes verification hard:
  undecidable in general, NP-complete for terminating programs
- Message Passing Interface (MPI): a de facto standard for C/C++

Example - I

P_0	P_1	P_2
$S_{0,0}(1)$	$R_{1,0}(*)$	$S_{2,0}(1)$
$B_{0,1}(0)$	$R_{1,1}(*)$	$B_{2,1}(0)$
$S_{0,2}(1)$	$W_{1,2}(h_{1,1})$	
	$B_{1,3}(0)$	
	$R_{1,4}(*)$	

Figure 1: Example Message Passing Program

P_i → process with rank i
$S_{i,k}(j)$ → non-blocking send from P_i to P_j at index k
$R_{i,k}(j)$ → non-blocking receive from P_j to P_i at index k
$R_{i,k}(*)$ → wildcard receive to P_i at index k
$W_{i,k}(h_{i,j})$ → blocking wait at index k for action at index j for process P_i
$B_{i,j}(k)$ → k^{th} barrier action from P_i at index j

Example - II

P_0	P_1	P_2
$S_{0,0}(1)$	$R_{1,0}(*)$	$S_{2,0}(1)$
$B_{0,1}(0)$	$R_{1,1}(*)$	$B_{2,1}(0)$
$S_{0,2}(1)$	$W_{1,2}(h_{1,1})$	
	$B_{1,3}(0)$	
	$R_{1,4}(*)$	

Figure 1: Example Message Passing Program

- **Trace:** sequence of matches allowed by MPI semantics
- **A complete trace:** $\tau_1 = \langle \{S_{0,0}, R_{1,0}\}, \{S_{2,0}, R_{1,0}\}, \{W_{1,2}\}, \{B_{0,1}, B_{1,3}, B_{2,1}\}, \{S_{0,2}, S_{1,4}\} \rangle$
- **A deadlocking trace:**
  if $R_{1,1}(*)$ was $R_{1,1}(0)$, then $\tau' = \langle \{S_{0,0}, R_{1,0}\} \rangle$ is a deadlocking trace
- MPI semantics are encoded in:
  a set of allowed matches $\mathbb{M}$, and a matches-before order $<_{mo}$

Methodology

Motivation - I

- Communication structure of real-world MPI programs:
  - can be decomposed into **independently-verifiable** epochs
  - has symmetries which lead to **redundancies in the search space** of traces
- Epochs can uncover local symmetries
- Redundant verification of repeated epochs can be avoided
- Symmetry breaking predicates can speed up the search

Motivation - II

P_0	P_1	P_2
$S_{0,0}(1)$	$R_{1,0}(*)$	$S_{2,0}(1)$
$B_{0,1}(0)$	$R_{1,1}(*)$	$B_{2,1}(0)$
$S_{0,2}(1)$	$W_{1,2}(h_{1,1})$	
	$B_{1,3}(0)$	
	$R_{1,4}(*)$	

Figure 1: Example Message Passing Program

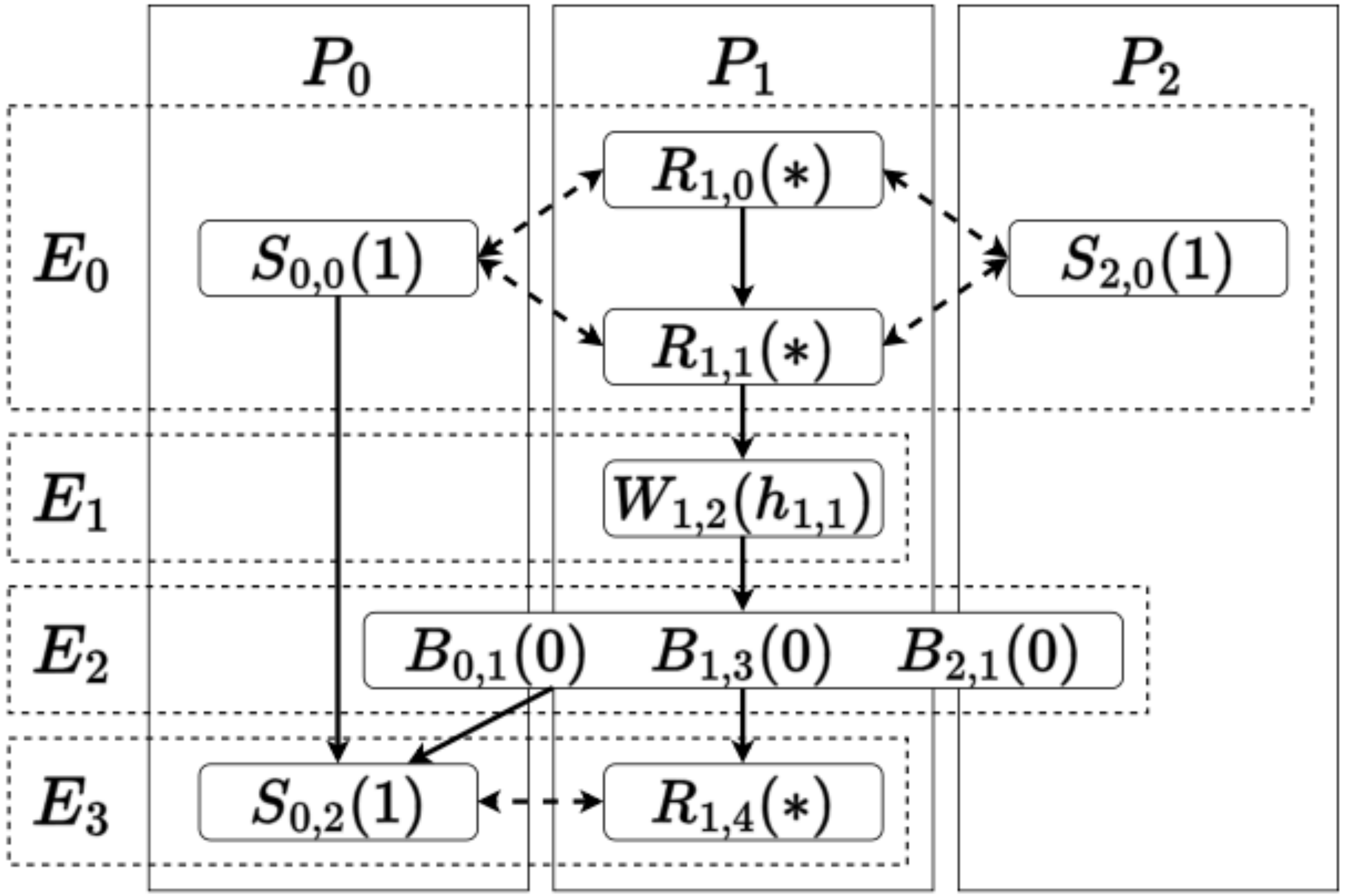


Figure 2: Program Graph for Running Example

Epoch Decomposition

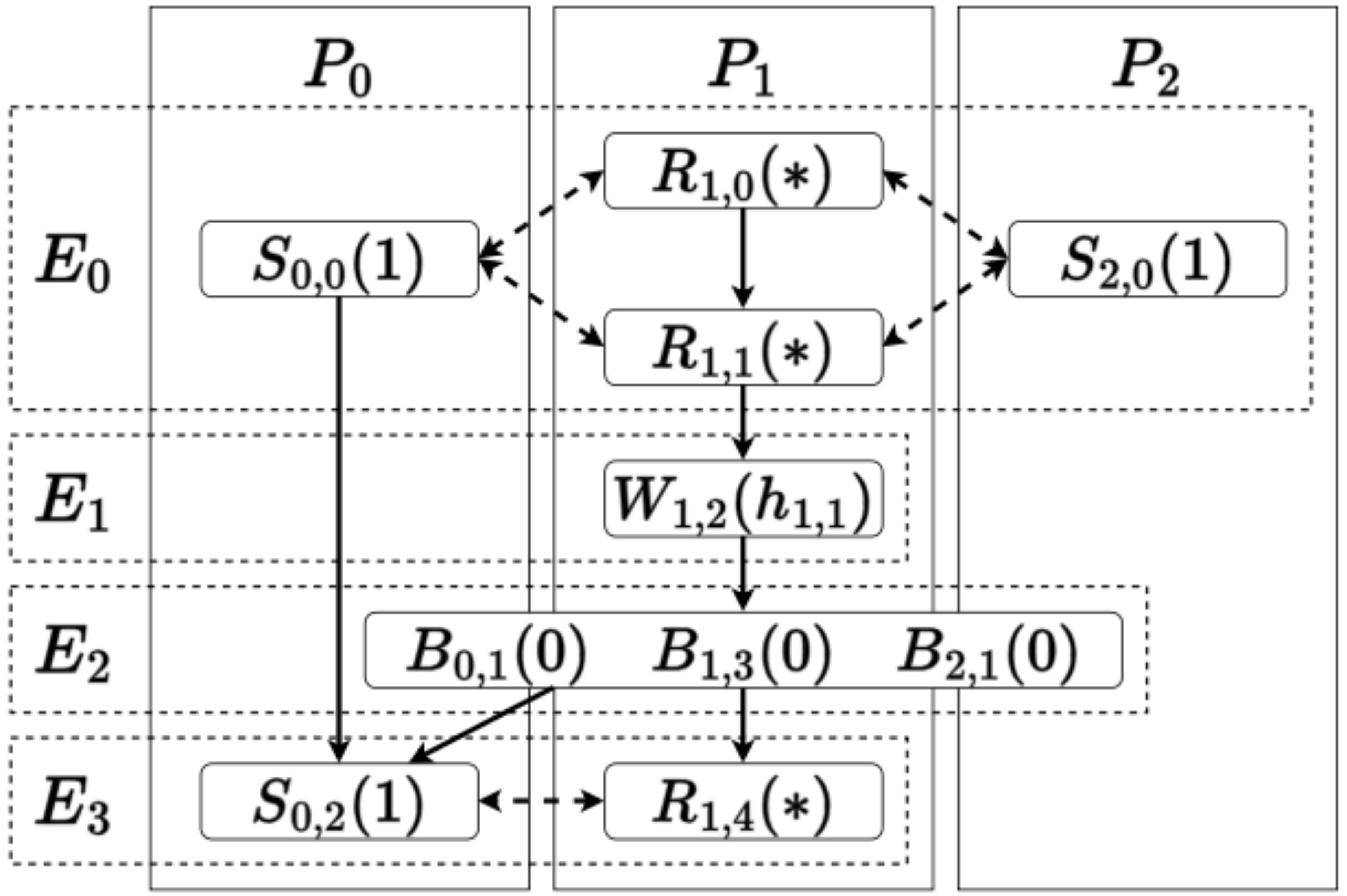


Figure 2: Program Graph for Running Example

- **Program Graph (PG):**
  nodes for actions,
  uni-directional edges (solid) for matches-before order,
  bi-directional edge (dashed) for potential matches
- **Epoch:** strongly connected component (SCC) of PG
- **Correctness:** THEOREM 4.3. *P has a deadlocking trace τ if and only if some communication epoch $e \in E$ has a deadlocking trace τ_e.*

Matchset refinement

- **Potential matches:** allowed matches which can be realized in some run
- Set of potential matches $\mathbb{M}^+$ is **NP-hard** to compute
- Correctness holds for any **over-approximation** M^+ of $\mathbb{M}^+$
- But efficacy relies on **tightness** of $\mathbb{M}^+$
- Start with $M^+ = \mathbb{M}$ (allowed matches), **refine** using pruning heuristics

Pruning Heuristics

- Recursive matches-before order pruning:
  for c_1 to match c_2, all ancestors of c_1 should find a match not successor of c_2
- Barrier-led pruning:
  pairs separated by a barrier cannot match
- Counting heuristic:
  if a set of Sends can match only a set of Recvs of same size,
  then this set of Recvs cannot match other Sends

Caching

- An epoch once verified is cached
- Epochs isomorphic to cached epochs are skipped
- BLISS package is used for graph isomorphism tests

Symmetry Detection

- Symmetry is captured by automorphisms of the epoch graph
- Example automorphism for epoch E_0:

$$\begin{aligned}\pi\colon\ & S_{0,0}(1) \mapsto S_{2,0}(1),\\ & S_{2,0}(1) \mapsto S_{0,0}(1),\\ & R_{1,0}(*) \mapsto R_{1,0}(*),\\ & R_{1,1}(*) \mapsto R_{1,1}(*).\end{aligned}$$

- Correctness: Traces equivalent under an automorphism are equivalent w.r.t. deadlock

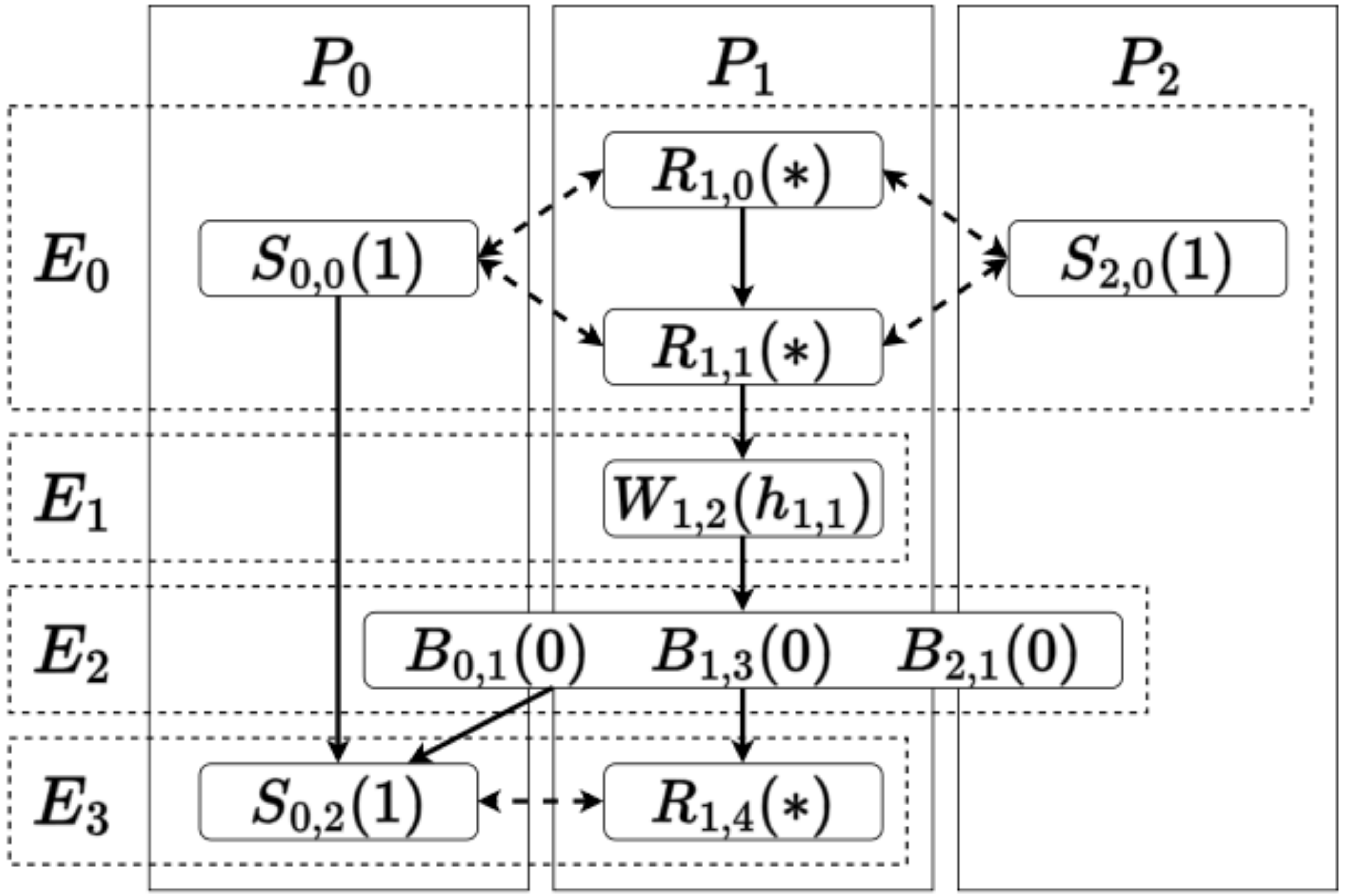


Figure 2: Program Graph for Running Example

Propositional Encoding

- $\phi_t(\tau) \rightarrow \tau$ is a valid trace
- $\phi_d(\tau) \rightarrow \tau$ is a deadlocking trace
- $\phi' = \phi_t \wedge \phi_d$ is SAT iff a deadlocking trace exists
- $\phi_s(\tau) \rightarrow$ satisfied by at least one representative from each equivalence class
- $\phi = \phi_t \wedge \phi_d \wedge \phi_s$ is SAT iff a deadlocking trace exists
- But ϕ is faster to solve, as symmetric branches are pruned from search space

SAT encoding

- The encoding for ϕ_t and ϕ_d are borrowed from prior work (Mopper)
- ϕ_s is encoded using Lex-Leader constraints
- Let $[\tau]$ denote a bit-vector for trace τ in the SAT formula
- For set B of generators of the automorphism group,

  $$\phi_s = \bigwedge_{\pi \in B} [\tau] \leq [\pi(\tau)]\text{, where } \leq \text{ is a lexicographic order}$$

- B is obtained using the BLISS package

Results

Implementation

- A prototype tool called **Simian** (https://github.com/rishabh-ranjan/simian)
- Components:
  - **Scheduler:** dynamic execution engine, borrowed from prior work (ISP)
  - **Analyzer:** epoch decomposition + symmetry breaking
  - **Solver:** verification of SAT formulas, we use the Z3 solver

Baselines

- **Mopper:** a SAT encoding for the entire program
- **Mopper-Opt:** alternative SAT encoding optimized for consecutive wildcards
- **Hermes:** a dynamic-symbolic hybrid verifier with an SMT formulation

Benchmarks

- **Adder:** adds an array of numbers, *master-worker* communication
- **Floyd:** all-pairs shortest path algorithm, *pipelined* communication
- **GaussElim:** Gauss-Jordan elimination, *pairwise* communication
- **Heat:** heat conduction simulation, *2D grid* topology
- **HeatErrors:** Heat, seeded with a *deadlock*
- **Integrate:** numerical integration using trapezoidal rule, *master-worker* communication
- **Diffusion:** iterative solver for diffusion, *2D grid* topology
- **MatMul:** implementation of matrix multiplication, *block-distributed communication over rows*

Variations

- **Diffusion** with fixed timesteps, but varying number of processes
- **Diffusion** on 2x2 grid, 4 processes, with varying timesteps
- **MatMul** on 8x8 matrices with varying number of processes
- **MatMul** with pxp matrices for p processes
- **MatMul** with fixed number of processes, but varying matrix size

Table 1: Runtimes for ∞-buffering (in s)

Name vs X	X	Deadlock	Mopper	Mopper-Opt	Hermes	Simian
Adder	8	No	0.268	**0.042**	0.212	0.061
vs	16	No	TO	**0.212**	TO	1.157
Processes	32	No	TO	**1.497**	TO	1.912
	64	No	TO	**4.116**	TO	7.257
Floyd	8	No	2.761	6.132	1.453	**0.628**
vs	16	No	283.524	399.156	2.148	**1.939**
Processes	32	No	TO	TO	5.021	**2.491**
	64	No	TO	TO	10.312	**6.081**
GaussElim	8	No	0.243	0.233	0.187	**0.186**
vs	16	No	0.628	1.655	0.258	**0.283**
Processes	32	No	4.314	4.282	1.993	**2.334**
	64	No	10.033	6.159	3.912	**3.226**
Heat	8	No	0.666	0.395	0.406	**0.325**
vs	16	No	1.581	0.845	0.636	1.506
Processes	32	No	6.543	1.623	2.005	4.255
	64	No	14.927	10.597	4.464	**3.232**
HeatErrors	8	Yes	0.523	0.395	**0.309**	0.353
vs	16	Yes	1.191	0.779	0.662	**0.565**
Processes	32	Yes	5.706	2.574	2.712	**2.191**
	64	Yes	10.392	**4.799**	6.435	5.051
Integrate	8	No	0.256	**0.038**	0.209	0.062
vs	16	No	TO	0.232	TO	**0.131**
Processes	32	No	TO	3.581	TO	**1.854**
	64	No	TO	**4.212**	TO	7.583
Diffusion	4	No	TO	3.729	239.77	**0.122**
(Timesteps=1)	8	No	TO	14.854	TO	**0.716**
vs	16	No	TO	49.069	TO	**8.566**
Processes	24	No	TO	271.247	TO	**36.308**
Diffusion	2	No	TO	1.598	TO	**0.225**
(Grid=2x2)	4	No	TO	26.347	TO	**1.023**
vs	8	No	TO	842.951	TO	7.197
Timesteps	16	No	TO	TO	TO	**67.854**
MatMul	8	No	2.719	0.083	1.815	**0.076**
(N=L=M=8)	16	No	3.572	0.274	3.032	**0.114**
vs	32	No	4.448	**0.734**	4.014	1.477
Processes	64	No	8.625	4.001	5.482	**2.766**
MatMul	8	No	2.708	0.086	1.918	**0.076**
(N=L=M=p)	16	No	TO	0.328	TO	**0.224**
vs	32	No	TO	3.728	TO	**2.822**
Processes	64	No	TO	**4.808**	TO	16.175
MatMul	4	No	0.061	**0.062**	0.071	0.065
(p=8)	6	No	0.101	**0.066**	0.112	0.079
vs	8	No	2.704	0.092	1.794	**0.076**
Size (N=L=M)	12	No	TO	9.772	TO	**4.938**

Table 2: Runtimes for 0-buffering (in s)

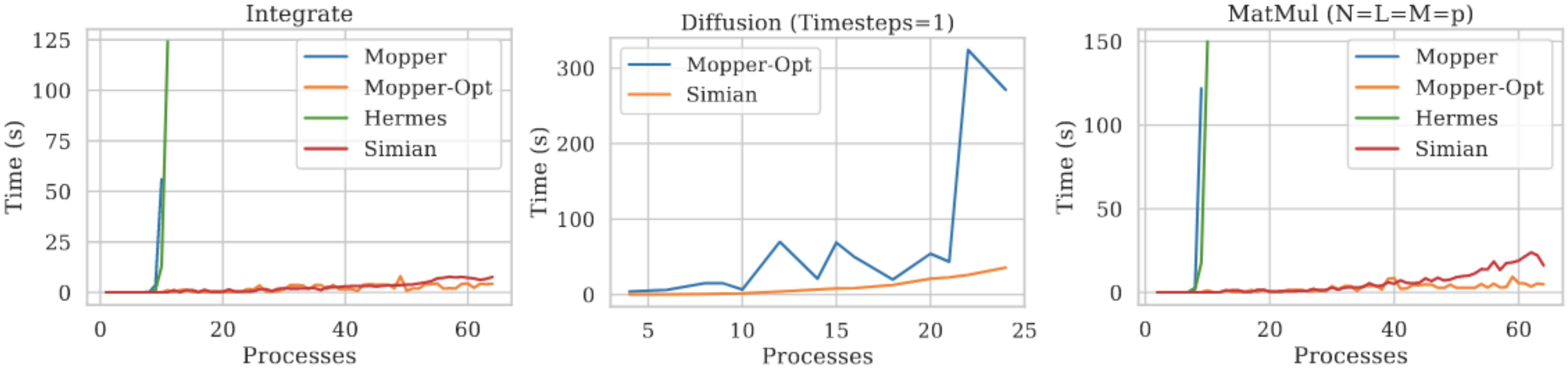
Name vs X	X	Deadlock	Mopper	Mopper-Opt	Hermes	Simian
Adder	8	No	0.281	**0.043**	0.285	0.061
vs	16	No	TO	**1.249**	TO	1.301
Processes	32	No	TO	1.684	TO	**0.932**
	64	No	TO	**4.112**	TO	7.065
Floyd	8	No	3.986	3.715	**0.232**	0.249
vs	16	No	7.171	22.413	**1.582**	2.108
Processes	32	No	112.793	157.188	**4.665**	5.619
	64	No	763.232	TO	**10.941**	32.261
GaussElim	8	No	0.223	0.235	**0.177**	0.185
vs	16	No	0.599	0.618	0.773	**0.481**
Processes	32	No	4.398	**1.297**	4.097	4.155
	64	No	5.748	6.085	5.612	**5.429**
Heat	8	No	0.323	0.325	0.241	**0.225**
vs	16	No	**0.726**	0.789	1.646	1.549
Processes	32	No	4.515	2.497	4.119	**1.831**
	64	No	4.139	**3.706**	5.341	9.121
HeatErrors	8	Yes	0.408	0.323	**0.221**	0.229
vs	16	Yes	1.226	1.772	0.587	**0.461**
Processes	32	Yes	4.351	2.515	4.568	**1.987**
	64	Yes	9.492	6.634	6.548	**2.932**
Integrate	8	No	0.281	**0.042**	0.251	0.079
vs	16	No	TO	**0.223**	TO	0.279
Processes	32	No	TO	**3.592**	TO	3.961
	64	No	TO	**4.237**	TO	9.119
Diffusion	4	Yes	**0.029**	0.032	0.115	0.137
(Timesteps=1)	8	Yes	0.037	**0.035**	0.132	0.125
vs	16	Yes	0.229	**0.197**	0.295	0.337
Processes	24	Yes	1.457	1.434	**0.548**	1.482
Diffusion	2	Yes	**0.028**	0.034	0.131	0.127
(Grid=2x2)	4	Yes	0.031	**0.028**	0.132	0.127
vs	8	Yes	**0.029**	0.035	0.121	0.112
Timesteps	16	Yes	**0.032**	0.038	0.124	0.139
MatMul	8	Yes	0.129	**0.069**	0.087	0.157
(N=L=M=8)	16	No	4.088	1.269	1.339	**0.242**
vs	32	No	5.097	**1.488**	3.714	3.582
Processes	64	No	7.544	4.065	**3.835**	4.209
MatMul	8	Yes	0.134	**0.073**	0.087	0.157
(N=L=M=p)	16	Yes	0.869	**0.281**	0.714	0.356
vs	32	Yes	4.649	**0.633**	9.545	1.169
Processes	64	Yes	24.643	**4.586**	98.515	9.554
MatMul	4	No	0.059	**0.046**	0.068	0.075
(p=8)	6	No	0.092	**0.048**	0.104	0.067
vs	8	Yes	0.127	**0.071**	0.098	0.157
Size (N=L=M)	12	Yes	0.189	**0.099**	0.121	0.187

Table 3: Communication Structure Summaries

Name vs X	X	Trace Size	Epochs				Symmetry
			(Size, Symmetry, Repeats) ...	Total	Unique	Repeated	Total
Adder	8	28	(14,6,1) (2,1,7)	8	2	6	7
vs	16	60	(30,14,1) (2,1,15)	16	2	14	15
Processes	32	124	(62,30,1) (2,1,31)	32	2	30	31
	64	252	(126,62,1) (2,1,63)	64	2	62	63
Floyd	8	176	(20,6,1) (147,0,1) (1,0,9)	11	3	8	6
vs	16	368	(20,6,1) (23,7,8) (147,0,1) (1,0,17)	27	4	23	13
Processes	32	752	(20,6,1) (23,7,24) (147,0,1) (1,0,33)	59	4	55	13
	64	1520	(20,6,1) (23,7,56) (147,0,1) (1,0,65)	123	4	119	13
GaussElim	8	84	(4,1,6) (2,1,2) (1,0,14)	22	3	19	2
vs	16	172	(4,1,14) (2,1,2) (1,0,22)	38	3	35	2
Processes	32	348	(4,1,30) (2,1,2) (1,0,38)	70	3	67	2
	64	700	(4,1,62) (2,1,2) (1,0,70)	134	3	131	2
Heat	8	144	(2,1,60) (1,0,24)	84	2	82	1
vs	16	296	(2,1,124) (1,0,48)	172	2	170	1
Processes	32	600	(2,1,252) (1,0,96)	348	2	346	1
	64	1208	(2,1,508) (1,0,192)	700	2	698	1
HeatErrors	8	144	(17,0,1) (2,1,31) (1,0,15)	47	3	44	1
vs	16	296	(33,0,1) (2,1,63) (1,0,31)	95	3	92	1
Processes	32	600	(65,0,1) (2,1,127) (1,0,63)	191	3	188	1
	64	1208	(129,0,1) (2,1,255) (1,0,127)	383	3	380	1
Integrate	8	28	(14,6,1) (2,1,7)	8	2	6	7
vs	16	60	(30,14,1) (2,1,15)	16	2	14	15
Processes	32	124	(62,30,1) (2,1,31)	32	2	30	31
	64	252	(126,62,1) (2,1,63)	64	2	62	63
Diffusion	4	88	(2,1,16) (18,8,2) (1,0,5)	23	3	20	9
(Timesteps=1)	8	188	(2,1,32) (42,20,2) (1,0,5)	39	3	36	21
vs	16	388	(90,44,1) (2,1,64) (90,44,1) (1,0,5)	71	4	67	89
Processes	24	588	(138,68,1) (2,1,96) (138,68,1) (1,0,5)	103	4	99	137
Diffusion	2	150	(2,1,32) (18,8,3) (1,0,8)	43	3	40	9
(Grid=2x2)	4	274	(2,1,64) (18,8,5) (1,0,14)	83	3	80	9
vs	8	522	(2,1,128) (18,8,9) (1,0,26)	163	3	160	9
Timesteps	16	1018	(2,1,256) (18,8,17) (1,0,50)	323	3	320	9
MatMul	8	54	(46,5,1) (1,0,8)	9	2	7	5
(N=L=M=8)	16	78	(2,1,7) (48,7,1) (1,0,16)	24	3	21	8
vs	32	126	(2,1,23) (48,7,1) (1,0,32)	56	3	53	8
Processes	64	222	(2,1,55) (48,7,1) (1,0,64)	120	3	117	8
MatMul	8	54	(46,5,1) (1,0,8)	9	2	7	5
(N=L=M=p)	16	110	(94,13,1) (1,0,16)	17	2	15	13
vs	32	222	(190,29,1) (1,0,32)	33	2	31	29
Processes	64	446	(382,61,1) (1,0,64)	65	2	63	61
MatMul	4	38	(2,1,3) (24,3,1) (1,0,8)	12	3	9	4
(p=8)	6	46	(2,1,1) (36,5,1) (1,0,8)	10	3	7	6
vs	8	54	(46,5,1) (1,0,8)	9	2	7	5
Size (N=L=M)	12	70	(62,5,1) (1,0,8)	9	2	7	5

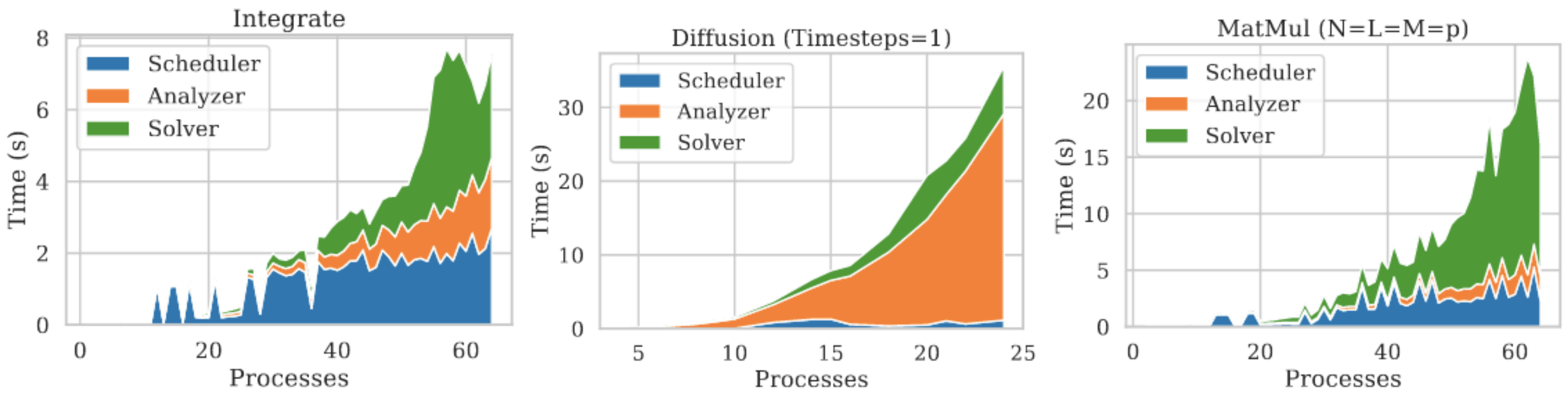
Table 4: Component-wise Times (in s)

Name vs X	X	Time		
		Scheduler	Analyzer	Solver
Adder vs Processes	8	**0.031**	0.014	0.013
	16	**1.084**	0.035	0.036
	32	**1.381**	0.206	0.238
	64	2.358	1.957	**2.938**
Floyd vs Processes	8	0.042	0.042	**0.543**
	16	**1.290**	0.094	0.553
	32	**1.708**	0.239	0.541
	64	**4.764**	0.753	0.558
GaussElim vs Processes	8	**0.151**	0.022	0.014
	16	**0.223**	0.032	0.014
	32	**2.273**	0.028	0.011
	64	**3.165**	0.045	0.011
Heat vs Processes	8	**0.264**	0.050	0.009
	16	**1.468**	0.031	0.006
	32	**4.195**	0.052	0.008
	64	**3.119**	0.099	0.007
HeatErrors vs Processes	8	**0.353**	0	0
	16	**0.565**	0	0
	32	**2.408**	0.073	0.016
	64	**3.407**	0.183	0.019
Integrate vs Processes	8	**0.032**	0.015	0.013
	16	**0.058**	0.035	0.037
	32	**1.372**	0.198	0.235
	64	2.682	1.972	**2.925**
Diffusion (Timesteps=1) vs Processes	4	0.039	**0.061**	0.022
	8	0.038	**0.585**	0.091
	16	0.135	**6.635**	1.783
	24	1.021	**27.953**	7.294
Diffusion (Grid=2x2) vs Timesteps	2	0.022	**0.185**	0.021
	4	0.027	**0.968**	0.021
	8	0.106	**6.952**	0.021
	16	0.982	**65.333**	0.021
MatMul (N=L=M=8) vs Processes	8	**0.032**	0.018	0.026
	16	**0.065**	0.024	0.023
	32	**1.389**	0.026	0.024
	64	**2.618**	0.032	0.022
MatMul (N=L=M=p) vs Processes	8	**0.032**	0.017	0.026
	16	0.066	0.043	**0.112**
	32	**1.699**	0.232	0.834
	64	2.331	2.109	**11.731**
MatMul (p=8) vs Size (N=L=M)	4	**0.032**	0.017	0.015
	6	**0.031**	0.018	0.021
	8	**0.032**	0.017	0.026
	12	0.034	0.028	**4.875**



(a) Integrate **(b) Diffusion (Timesteps=1)** **(c) MatMul (N=L=M=p)**

Figure 3: Total Times



(a) Integrate

(b) Diffusion (Timesteps=1)

(c) MatMul (N=L=M=p)

Figure 4: Component Times

Future Directions

- Extension to **multi-path programs** where communication affects control flow
- Applicability of our techniques to **other concurrency frameworks**
- Expanding the **scope of our tool** and evaluating against recent benchmarks

Thank you!
Questions?