# $\lambda_{\texttt{MPI}}$ : A Language Model for MPI

Keith Allen [1] Andrew K. Hirsch [1]

[1]University at Buffalo, SUNY

University at Buffalo
The State University of New York

## Introduction

The Message Passing Interface (MPI) [1] is the standard for high-performance computing (HPC) applications, ensuring correct communication in massively parallel systems. However, since MPI is often implemented in imperative languages, programs are inherently hard to reason about. We present $\lambda_{\texttt{MPI}}$, a new Concurrent Lambda Calculus to formalize the semantics of MPI.

## System Overview

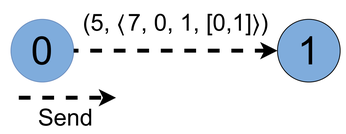
System Level:

- **Program** $\rho$ is a set number of named **processes** $p_n$, each running an **expression** $e_n$ in parallel.
- **State** $\sigma$ is a set of FIFO queues between each ordered pair of nodes $(p_n p_m)$.
- A **configuration** $\langle \rho, \sigma \rangle$ captures full system.

Local Level:

- Processes know only their expression.
- **Labelled transition system (LTS)** forms operational semantics.

## Point to Point Example




$p_0$ sends $5$ to $p_1$

$$p_0 \triangleright \texttt{send}\ 5\ \texttt{tag}\ 7\ \texttt{to}\ 1\ \texttt{of}\ {}^{\texttt{G}}[0,1]^{i_1} \xrightarrow{5 \rightsquigarrow \langle 7,0,1,{}^{\texttt{G}}[0,1]^{i_1} \rangle} ()$$

$$p_1 \triangleright \texttt{recv}\ x\ \texttt{in}\ x\ \texttt{select}\ \langle \texttt{tag}\ 7, \texttt{from}\ 0 \rangle\ \texttt{of}\ {}^{\texttt{G}}[0,1]^{i_2} \xrightarrow{\langle 7,0,1,5 \rangle \rightsquigarrow {}^{\texttt{G}}[0,1]^{i_1}} 5$$

Envelopes: $\langle tag, source, dest, group \rangle$

- All information of where a message need to go.
- Components appear in expressions, but full envelopes are only found in labels.

$\lambda_{\texttt{MPI}}$ supports **blocking** and **non-blocking sends** and **receives**.

## Collective Communication Example

$p_6 \triangleright$
$\texttt{gather}\ 1\ \texttt{from}$
$[\ \texttt{scatter}\ 1\ \texttt{of}\ [0;0;0;0]^i$
$\texttt{as}\ x\ \texttt{in}\ (x + (\texttt{rank}\ \mathbb{G}^i))$
$\texttt{to}\ {}^{\texttt{G}}[6,3,0,2]^i\ ]^j$
$\texttt{and}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{[[0]^i;[0]^i;[0]^i]^i \rightsquigarrow_{\texttt{sc}} {}^{\texttt{G}}[6,3,0,2]^i}$$

$\texttt{gather}\ 1\ \texttt{from}$
$[\ 0 + (\texttt{rank}\ \mathbb{G}^i)\ ]^j$
$\texttt{and}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{\texttt{rank}(\mathbb{G}^i) \rightsquigarrow 6}$$

$\texttt{gather}\ 1\ \texttt{from}\ [6]^j$
$\texttt{and}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{[[3]^i;[0]^i;[2]^i]^i \rightsquigarrow_{\texttt{ga}} {}^{\texttt{G}}[6,3,0,2]^i}$$

$[6;3;0;2]^i$

$p_3 \triangleright$
$\texttt{send}$
$[\ \texttt{recv}\ x\ \texttt{in}$
$(x + (\texttt{rank}\ \mathbb{G}^i))$
$\texttt{select}\ \langle \texttt{tag}\ *, \texttt{from}\ 0 \rangle$
$\texttt{of}\ {}^{\texttt{G}}[6,3,0,2]^i\ ]^j$
$\texttt{tag}\ 0\ \texttt{to}\ 0\ \texttt{of}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{\langle 0,0,1,{}^{\texttt{G}}[6,3,0,2]^i \rangle \rightsquigarrow [0]^j}$$

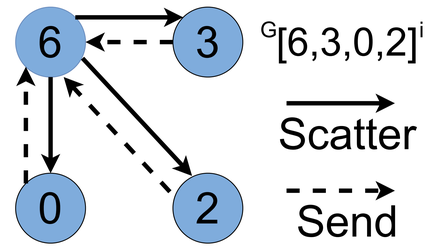
$\texttt{send}\ [(0 + (\texttt{rank}\ \mathbb{G}^i))]^j$
$\texttt{tag}\ 0\ \texttt{to}\ 0\ \texttt{of}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{\texttt{rank}(\mathbb{G}^i) \rightsquigarrow 3}$$

$\texttt{send}\ [3]^j$
$\texttt{tag}\ 0\ \texttt{to}\ 0\ \texttt{of}\ {}^{\texttt{G}}[6,3,0,2]^i$

$$\xrightarrow{[3]^j \rightsquigarrow_3 \langle 0,1,0,{}^{\texttt{G}}[6,3,0,2]^i \rangle}$$

$()$




All Processes:

- Receive the value $0$ from the root.
- Compute their rank in ${}^{\texttt{G}}[6,3,0,2]^i$.
- Returns their rank added to the $0$.

Collective Communication:
$\lambda_{\texttt{MPI}}$ supports MPI **Broadcast**, **Scatter**, and **Gather** as primitives as well as **Reduce** and **Scan**.

## Selection of Syntax and Stepping Rules

$$\begin{array}{lll}
\text{Expressions } e & ::= x \mid \tilde{n} \mid () \mid \texttt{error}(i) \mid * \mid \langle \texttt{tag}\ e_1, \texttt{from}\ e_2 \rangle \mid \texttt{send}\ e\ \texttt{tag}\ n\ \texttt{to}\ p\ \texttt{of}\ g \mid \texttt{recv}\ x\ \texttt{in}\ e_1\ \texttt{select}\ e_2\ \texttt{of}\ g \\
& \quad \mid \texttt{scatter}\ n\ \texttt{of}\ e_1\ \texttt{as}\ x\ \texttt{in}\ e_2\ \texttt{to}\ g \mid \texttt{gather}\ n\ \texttt{from}\ e\ \texttt{and}\ g \mid [e_0;\ldots;e_n]^i \mid {}^{\texttt{G}}[e_0,\ldots,e_n]^i \mid \texttt{rank}\ g \\
\text{Values} & v ::= () \mid \tilde{n} \mid \texttt{error}(i) \mid * \mid \langle \texttt{tag}\ v_1, \texttt{from}\ v_2 \rangle \mid [v_1;\ldots;v_n]^i \mid \mathbb{G}^i \mid \emptyset^i \mid {}^{\texttt{G}}[n]^i \mid {}^{\texttt{G}}[n_0,\ldots,n_m]^i \\
\text{Labels} & l ::= \epsilon \mid v \rightsquigarrow \langle t,p,q,g \rangle \mid \langle t,p,q,g \rangle \rightsquigarrow v \mid v \rightsquigarrow_{\texttt{br}} g \mid v \rightsquigarrow_{\texttt{sc}} g \mid g \rightsquigarrow_{\texttt{ga}} v \mid \texttt{rank}(g) \rightsquigarrow n \\
\text{Programs} & \rho ::= p_1 \triangleright e_1 \| \ldots \| p_n \triangleright e_m
\end{array}$$

$$\text{Send}\ \frac{}{\texttt{send}\ v\ \texttt{tag}\ t\ \texttt{to}\ q\ \texttt{of}\ g \xrightarrow{v \rightsquigarrow \langle t,\texttt{rank}\ g,q,g \rangle} ()} \qquad \text{Recv}\ \frac{}{\texttt{recv}\ x\ \texttt{in}\ e\ \texttt{select}\ \langle \texttt{tag}\ t_1, \texttt{from}\ p_1 \rangle\ \texttt{of}\ g \xrightarrow{\langle t_2,p_2,q,g \rangle \rightsquigarrow v} e[x \mapsto v]}$$

$$\text{Scatter}\ \frac{(n_2+1) = n_1(n_3+1) \qquad (\forall b.1 \le b \le n_3 \wedge a_b = [v_{bn_1};\ldots;v_{(b+1)n_1-1}]^c, \text{where } c \text{ is fresh})}{\texttt{scatter}\ n_1\ \texttt{of}\ [v_0;\ldots;v_{n_2}]^{i_1}\ \texttt{as}\ x\ \texttt{in}\ e\ \texttt{to}\ {}^{\texttt{G}}[p_0,\ldots,p_{n_3}]^{i_2} \xrightarrow{[a_1;\ldots;a_{n_3}]^{i_3} \rightsquigarrow_{\texttt{sc}} {}^{\texttt{G}}[p_0,\ldots,p_{n_3}]^{i_2}} e[x \mapsto [v_1;\ldots;v_{(n_1-1)}]^{i_4}]}$$

$$\text{PSend}\ \frac{g = {}^{\texttt{G}}[x_0,\ldots,x_n]^i \qquad 0 \le p,q \le n \qquad \rho(p_{x_p}) = e_{x_p} \qquad \rho(p_{x_q}) = e_{x_q} \qquad e_{x_p} \xrightarrow{v \rightsquigarrow \langle t_1,p,q,g \rangle} e'_{x_p} \qquad e_{x_q} \xrightarrow{\langle t_2,p_2,q,g \rangle \rightsquigarrow v} e'_{x_q} \qquad \texttt{true} = \texttt{tmatch}(t_1,t_2) \qquad \texttt{true} = \texttt{rmatch}(p,p_2) \qquad \sigma(\texttt{peek}\ p_{x_p} p_{x_q}) = \bot}{\langle \rho,\sigma \rangle \to_\rho \langle \rho[p_{x_p} \mapsto e'_{x_p}, p_{x_q} \mapsto e'_{x_q}], \sigma \rangle}$$

$$\text{PSc}\ \frac{g = {}^{\texttt{G}}[x_0,x_1,\ldots,x_n]^{i_1} \qquad \rho(p_{x_0}) = e_{x_0} \qquad e_{x_0} \xrightarrow{[v_1;\ldots;v_n]^{i_2} \rightsquigarrow_{\texttt{sc}} g} e'_{x_0} \qquad (\forall m.\, 1 \le m \le n \wedge \rho(p_{x_m}) = e_{x_m} \wedge e_{x_m} \xrightarrow{\langle t,0,m,g \rangle \rightsquigarrow v_m} e'_{x_m} \wedge \texttt{peek}\ p_{x_0} p_{x_m} = \bot)}{\langle \rho,\sigma \rangle \to_\rho \langle \rho[p_{x_m} \mapsto e'_{x_m} | \forall m.0 \le m \le n], \sigma \rangle}$$

## Group Operations

MPI Groups:

- Logical collections of processes.
- Each process $p$ has a **rank** $0$ to $(n-1)$.
- Rank $0$ is the **root** of a group.
- Written as lists of **global ranks**, index of the rank being the **local rank**.

Default groups: $n$ processes and $0 \le m < n$

- Global Group: $\mathbb{G}^i = {}^{\texttt{G}}[0;\ldots;(n-1)]^i$
- Empty Group: $\emptyset^i = {}^{\texttt{G}}[\,]^i$
- Singleton Group: ${}^{\texttt{G}}[m]^i$

With the default groups $\lambda_{\texttt{MPI}}$ can construct all valid groups, such as ${}^{\texttt{G}}[6;3;0;2]^i$, where global rank $6$ has local rank $0$.

Group Constructors:

- $\texttt{gunion}\ {}^{\texttt{G}}[6;3;0;2]^{i_1}\ {}^{\texttt{G}}[1;2;5;3]^{i_2} \xrightarrow{\epsilon} {}^{\texttt{G}}[6;3;0;2;1;5]^{i_3}$
- $\texttt{gsplit}\ (\texttt{fun}\ f\ x := x \mod 2)\ {}^{\texttt{G}}[6;3;0;2]^{i_1} \xrightarrow{\epsilon} [{}^{\texttt{G}}[6;0]^{i_3}; {}^{\texttt{G}}[3;2]^{i_4}]^{i_2}$
- $\texttt{gshuffle}\ [3;0;2;1]^{i_1}\ {}^{\texttt{G}}[6,3,0,2]^{i_2} \xrightarrow{\epsilon} {}^{\texttt{G}}[2,6,0,3]^{i_3}$

$\lambda_{\texttt{MPI}}$ also supports other MPI group constructors, such as **intersection** and **difference**, using these three defined operations.

## Future Work

- Formalize $\lambda_{\texttt{MPI}}$ in the Rocq proof assistant.
- Use $\lambda_{\texttt{MPI}}$ as the target language in a choreographic [2] HPC language based on MPI.
- Explore program analysis and verification using this choreographic language.

## References

1 Message Passing Interface Forum. 2025. MPI: A Message-Passing Interface Standard Version 5.0. https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf

2 Fabrizio Montesi. 2022. Introduction to Choreographies. Cambridge University Press. https://doi.org/10.1017/9781108981491

Contact: ktallen@buffalo.edu