

Collective Communication Support for Grid Computing

Thilo Kielmann
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

Joint work with
the Albatross project team members



High-Performance Computing on Grids



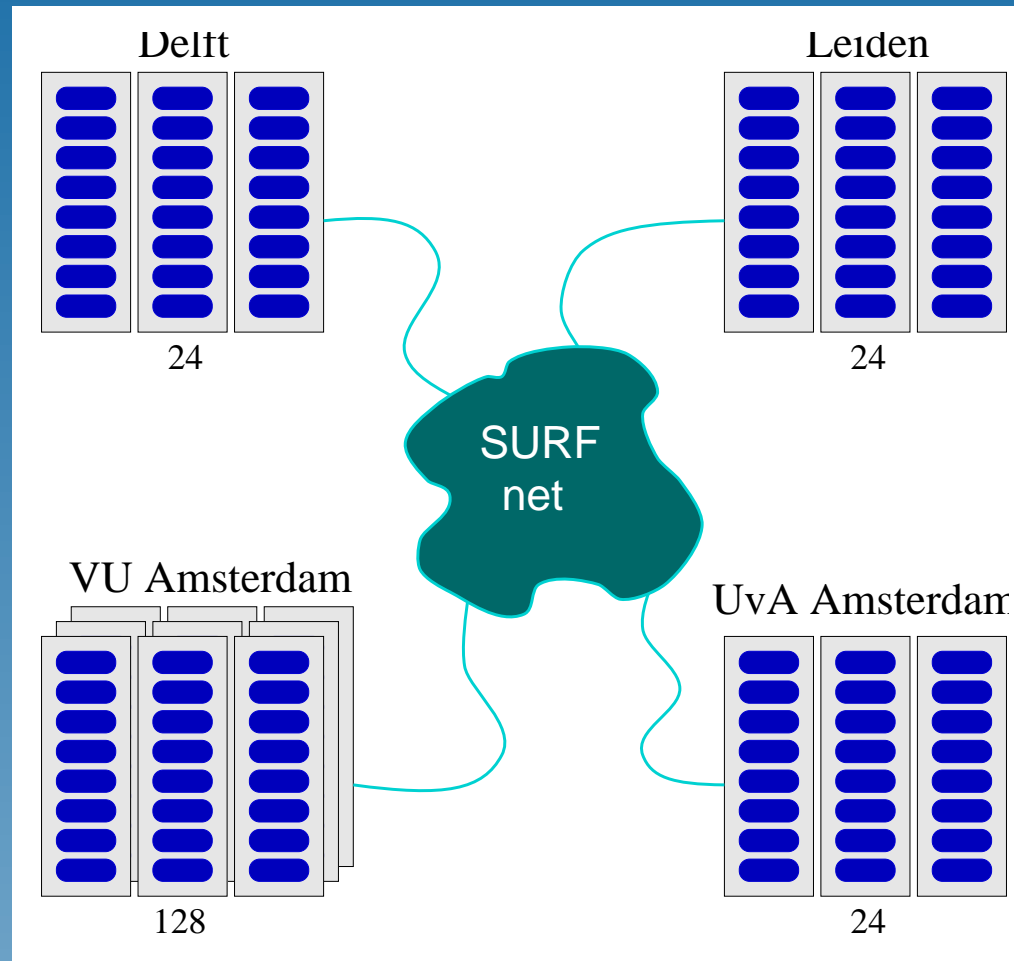
Collective Communication

- Common communication patterns in parallel apps
 - ★ broadcast
 - ★ reduction
 - ★ data (re-)distribution
- Building blocks for Grid applications
 - ★ need Grid-aware implementations

Talk Outline

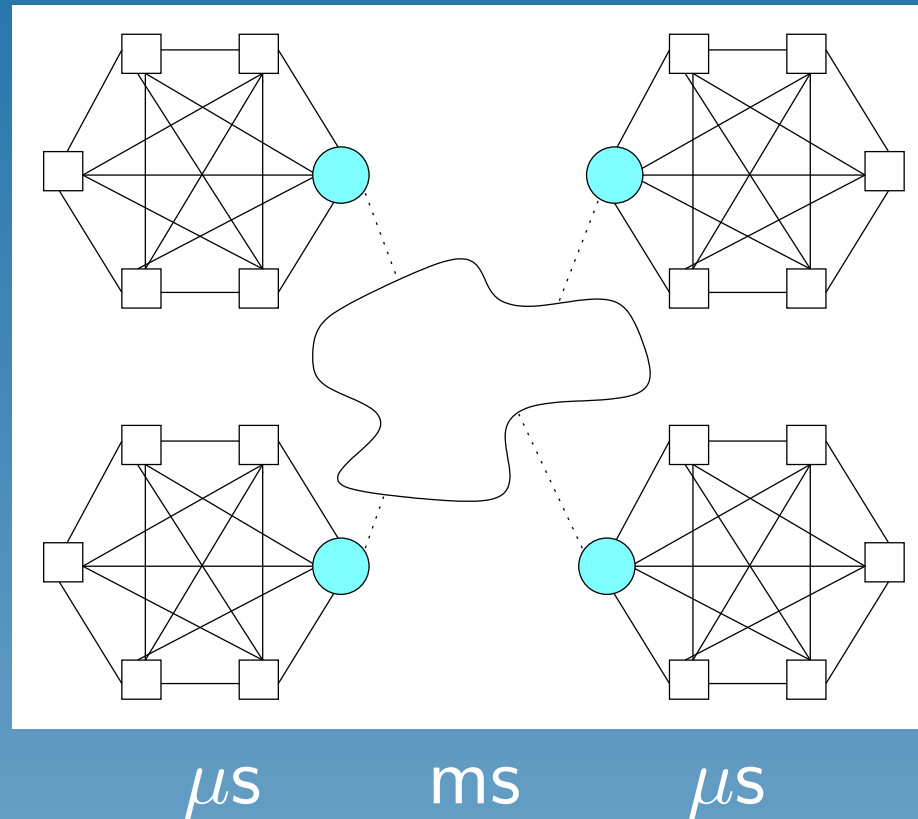
1. MagPie:
implementing collective communication for Grids
2. Group Method Invocation (GMI):
collective communication for Java (for Grids)

The “Dutch Grid”: the DAS



follow-up system (including Utrecht) operative mid 2001

Collectives for Grid Platforms



Current work assumption: WAN is fully connected

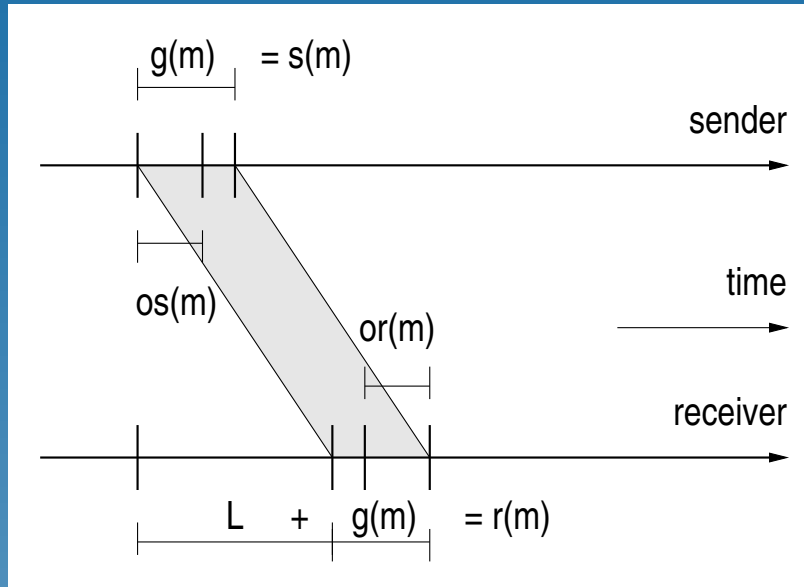
MagPle

Designing MPI's collective operations with minimal completion time for clustered Grid systems

MagPle in detail:

- The performance model P-LogP
- Basic collective operations
- Vision: MagPle for irregular networks

Performance Model: $P\text{-Log}P$



$m =$ message size

L latency

os send overhead

or receive overhead

g gap

P processors

transit time for single bit

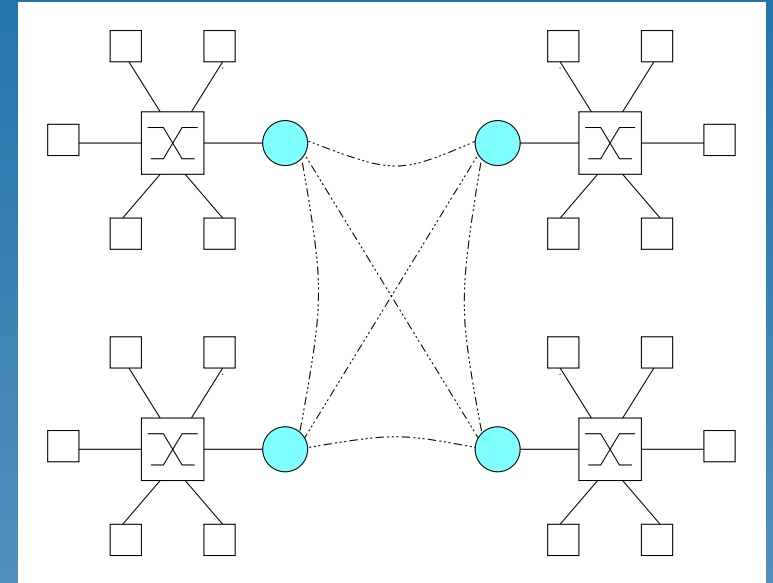
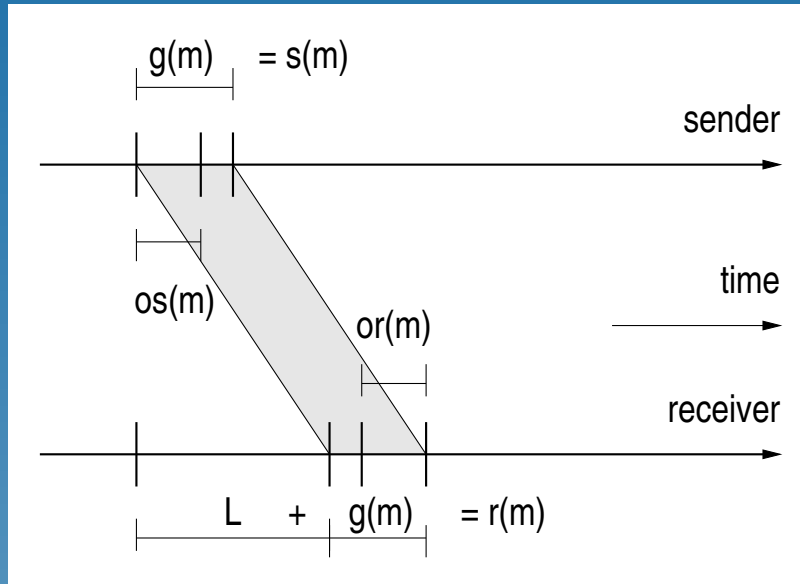
time, sender is busy with msg

time, receiver is busy with msg

time a msg “occupies” a link

(subscripts l and w for LAN and WAN)

Modeling Message Transmission



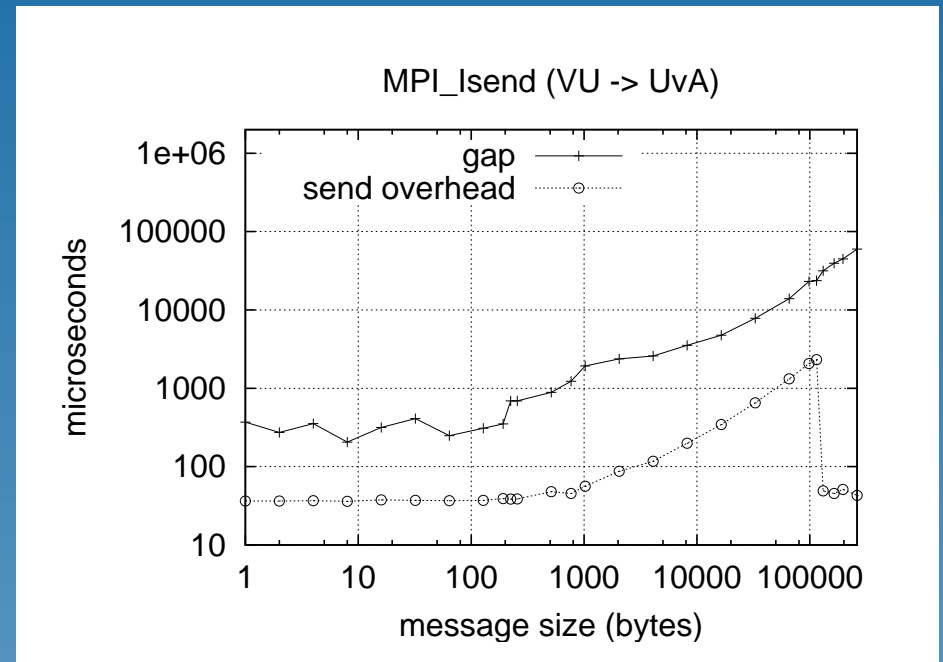
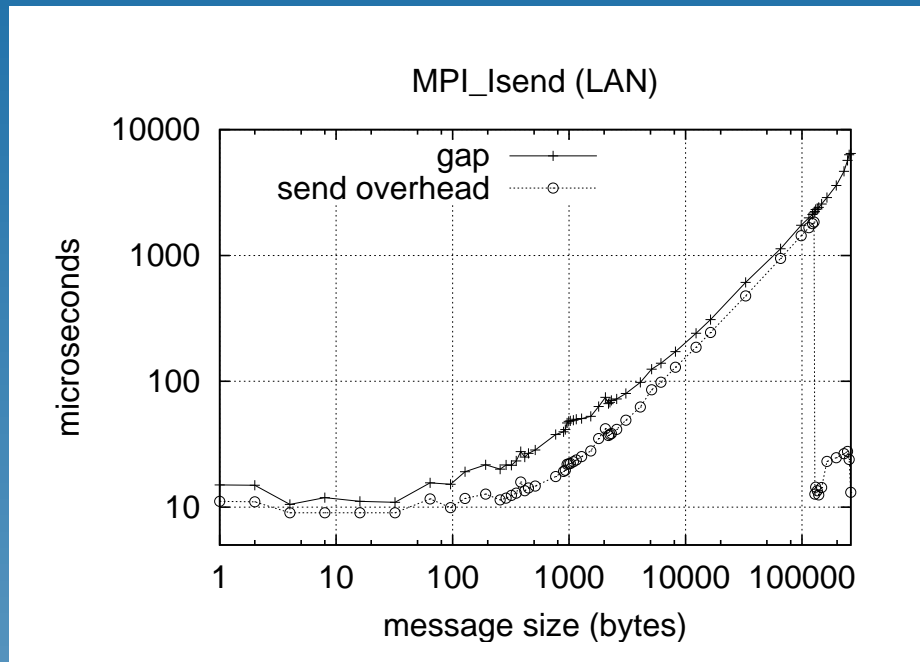
$s(m)$: sender is ready to send the next message

$r(m)$: receiver completely received the message

$s_l(m) = g_l(m)$ $r_l(m) = L_l + g_l(m)$

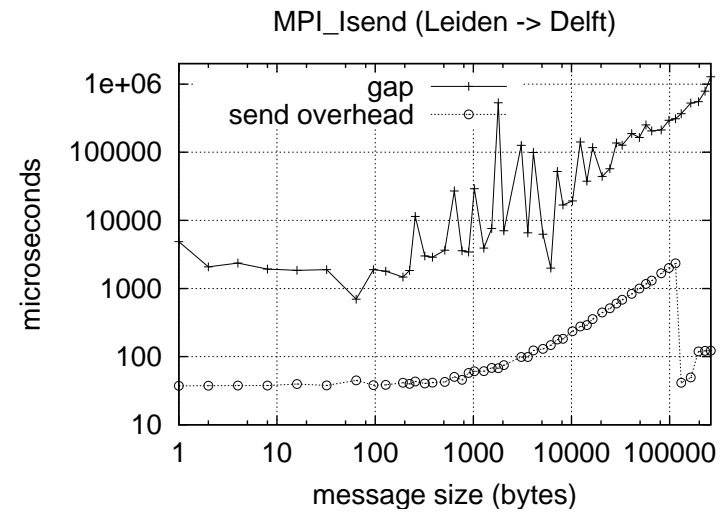
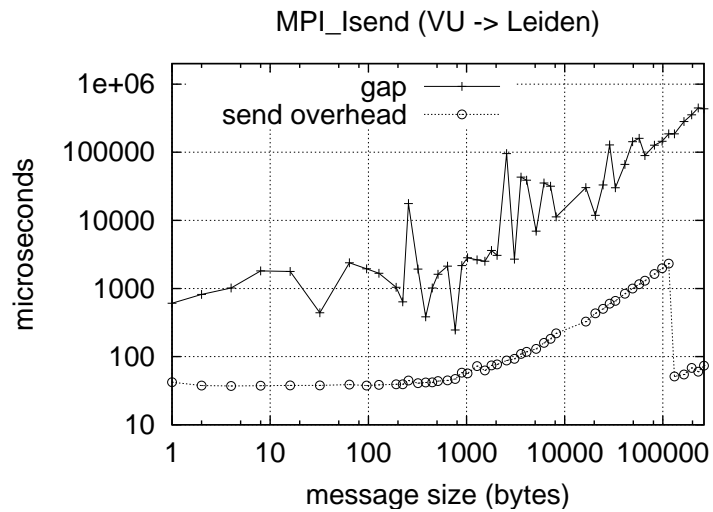
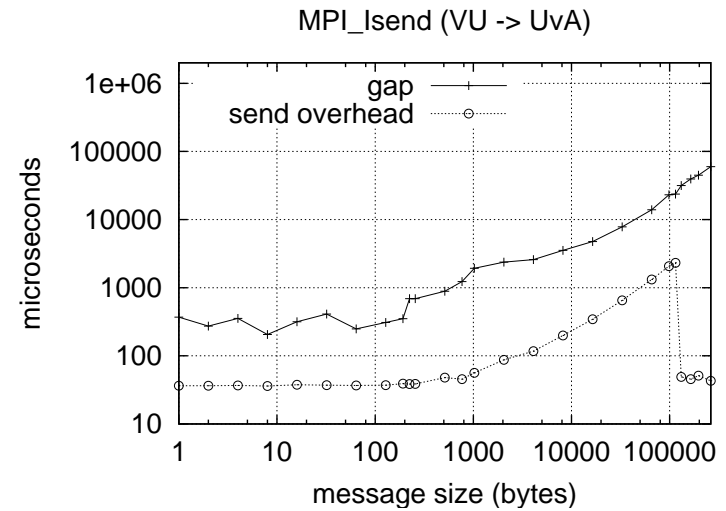
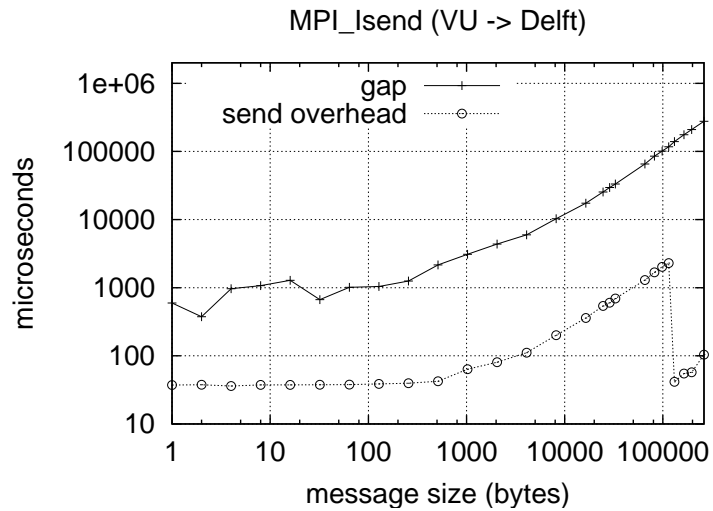
$s_w(m) = \max(g_l(m), os_w(m))$ $r_w(m) = L_w + g_w(m)$

Send Overhead and Gap on LAN and WAN



Observation: $os_w \ll g_w$ and $g_l \ll g_w \Rightarrow$ spare LAN cycles

Idea: Send small message segments quasi-simultaneously over many/all WAN links.

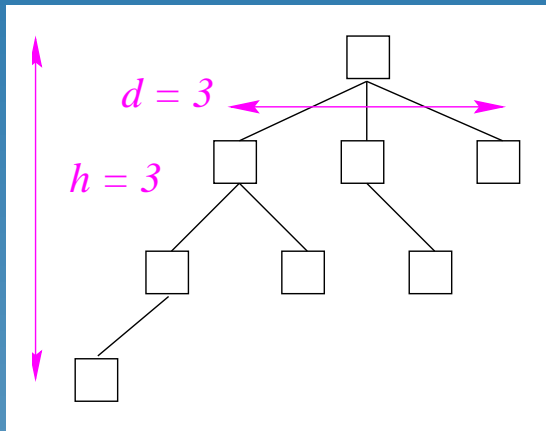


Basic Collective Operations

- Broadcast
- Scatter
- Gather
- Allgather

Broadcast

(Single layer: LAN only or WAN only)



M

message size

m

segment size

$k = \lceil M/m \rceil$

nbr of segments

d

degree of tree

h

height of tree

$$T = (k - 1) \cdot \gamma + \lambda$$

γ gap of tree
 λ latency of tree

binomial tree: ($d = \log P$, $h = \log P$)

$$\gamma \leq \max (g(m), or(m) + d \cdot s(m)) \quad \lambda \leq h \cdot ((d - 1) \cdot s(m) + r(m))$$

Optimizing Broadcast Trees

given: message size M and LAN/WAN parameters

sought: tree shapes (d_w and d_l), segment size m

$$T = (k - 1) \cdot \gamma + \lambda_w + \lambda_l \quad k = \lceil M/m \rceil$$

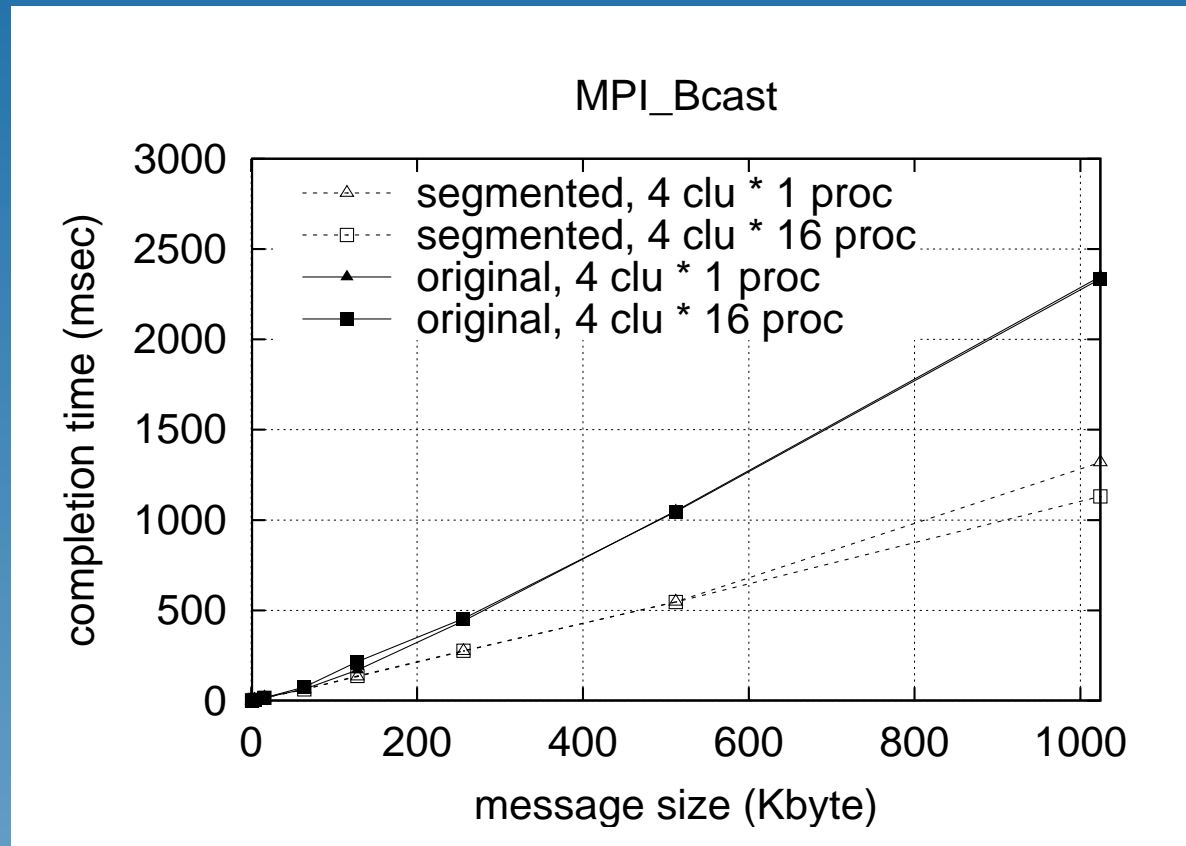
T depends on M

⇒ optimization at run time

⇒ heuristic search for m

(binary search + hill climbing)

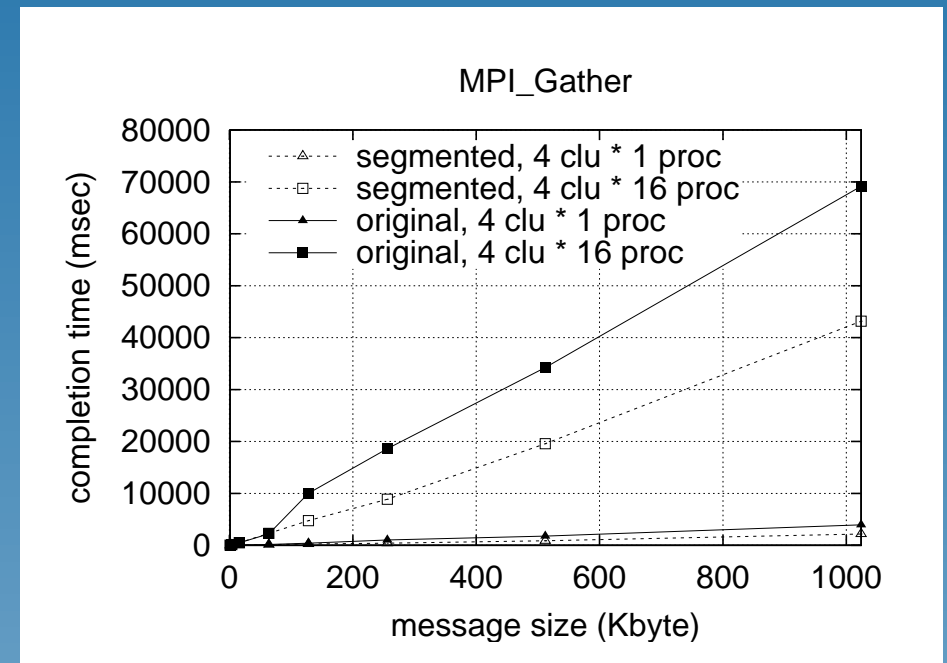
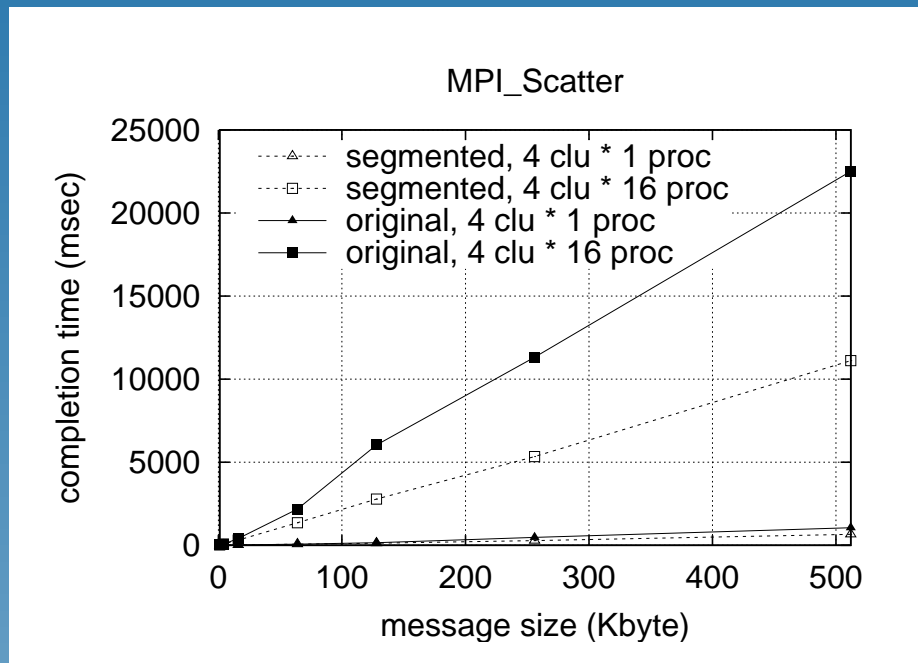
Experimental Results



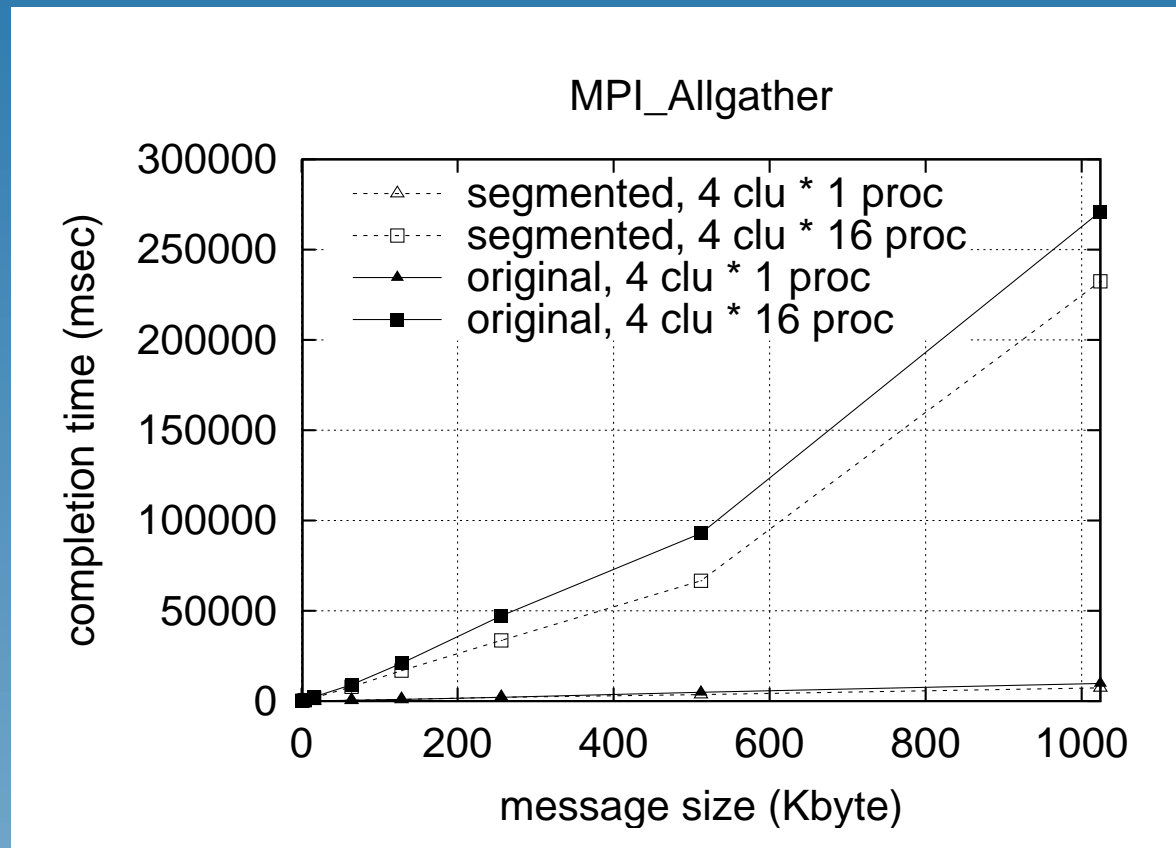
segmentation [Parallel Computing'01]

original MagPle [PPoPP'99]

Experimental Results (2)



Experimental Results (3)

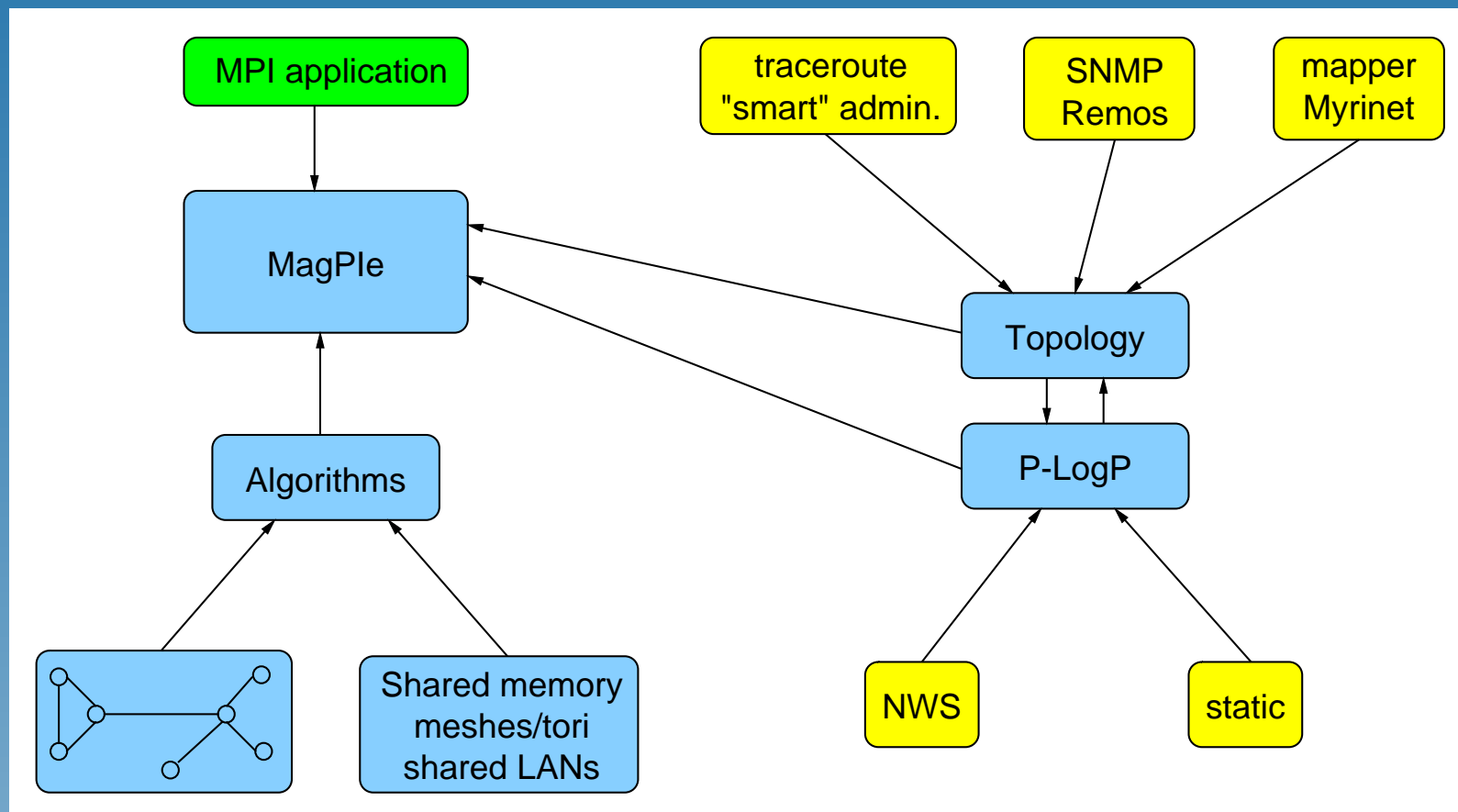


Summary (MagPie)

- P-LogP covers the performance aspects needed for modeling collective operations
- We can build collectives that efficiently use WAN links
 - ★ message segmentation
 - ★ communication graph shape optimization
- We can measure the P-LogP parameters efficiently
- The MagPie work is about *implementation* of collectives

Vision: MagPle for Irregular Networks

joint work with G. Fagg (UTK) and P. Geoffray (Myricom)



Group Method Invocation (GMI)

- Albatross is working on a Java-based platform for Grid computing:
 - ★ Java is ubiquitous
 - ★ Java is becoming faster (getting in the order of C)
 - ★ Java allows object-oriented communication operations
- This work is about *design* of collectives

Adding Collective Communication to Java

1. MPJ: Java language binding of MPI
 - + uses existing MPI-library (efficient)
 - + well-known API
 - does not fit into Java object model
 - ★ processes instead of threads
 - ★ arrays instead of objects
 - ★ no polymorphism

Adding Collective Communication to Java

2. CCJ: MPI-like library on top of RMI

- + supports polymorphism
- + supports objects and threads
- still does not fit into the method-invocation model
- hard to implement efficiently

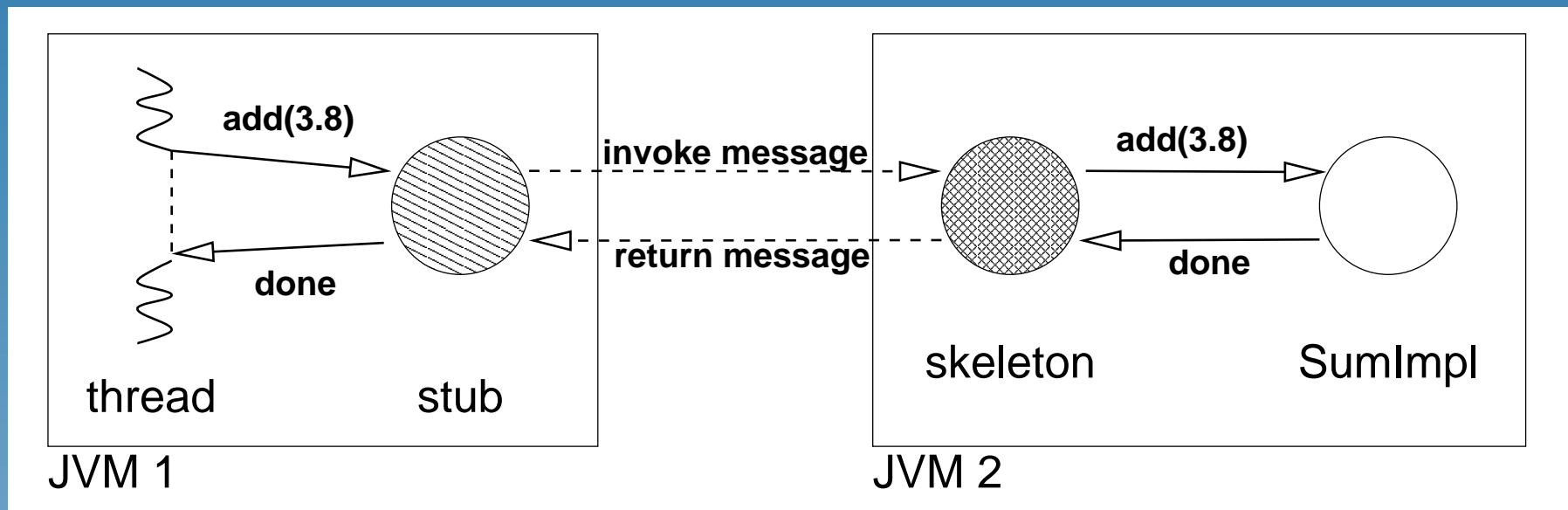
Adding Collective Communication to Java

3. Group Method Invocation (GMI)

- + uses method invocation model (with polymorphism)
- + based on object groups (rather than threads)
- + subsumes local MI, RMI, and GMI
- + allows the use of efficient low-level communication underneath

Remote Method Invocation (RMI)

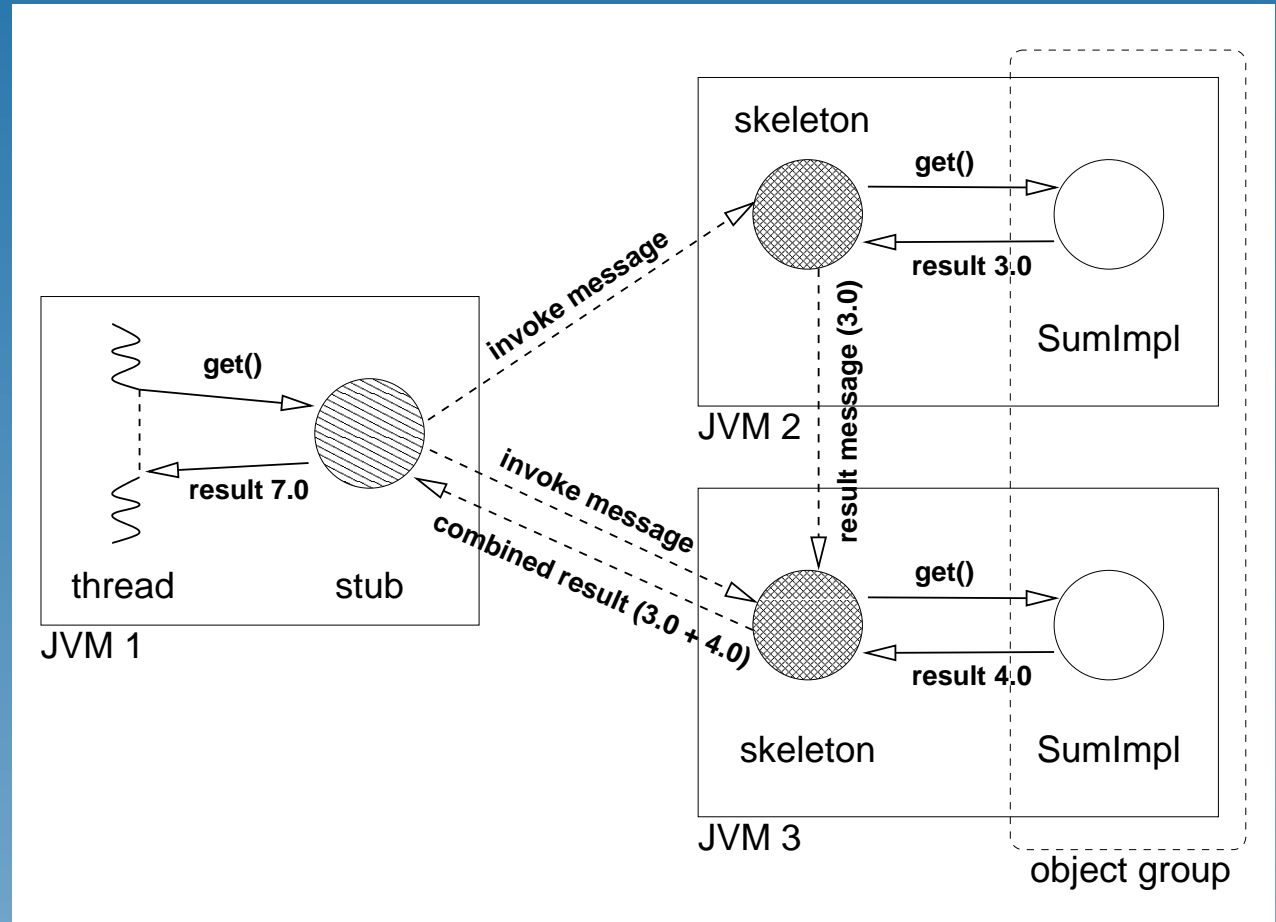
- Remote object implements *Remote Interface*
- Compiler (*rmic*) generates stub and skeleton



Group Method Invocation (GMI)

Group object
implements
Group Interface

Compiler
generates stub
and skeleton



GMI Model

- Groups are created dynamically
 - ★ immutable after creation
 - ★ ordered (MPI-style size and rank)
- Stubs can be configured to forward methods to:
local object / remote object /
entire group / entire group, personalized
- Skeletons can be configured to treat result:
discard / return one / combine all

Methods in GMI: Orthogonal Combination

invocation	result		
	discard	return (one)	combine
local	async. invocation	std. invocation	MPI-style combine
remote	async. RMI	RMI	MPI-style combine
group	async. multicast	multicast	group combine
personal.	async. scatter	scatter	personal. combine

Configuration via reflection (not time-critical):

- `Group.setInvoke(object, "double get()", GROUP)`
- `Group.setResult(object, "double get()", COMBINE, "method")`

GMI Implementation

- Prototype based on Manta (native Java compiler)
 - ★ speed competitive with mpiJava and direct RMI code
- Work in progress:
 - ★ Implementation JVM-independent (pure Java)
 - ★ Uses new communication library: *Ibis*
 - ★ op top of TCP, RMI, Manta, . . .

Conclusions

- Collective communication provides important building blocks for high-performance applications on Grids
- MagPIe provides efficient implementation
- GMI provides seamless integration to Java model
- MagPIe and GMI still need to be integrated





<http://www.cs.vu.nl/albatross/>