Experimental Evaluation of BSP Programming Libraries

Peter Krusche

Department of Computer Science
Centre for Scientific Computing
University of Warwick
peter@dcs.warwick.ac.uk

Outline

Motivation...

Study and compare the communication characteristics and performance predictability of ‘BSP-style’ communication libraries.

Outline

1. The BSP Model
2. BSP Programming Libraries
3. Benchmarking
4. Performance and Predictability

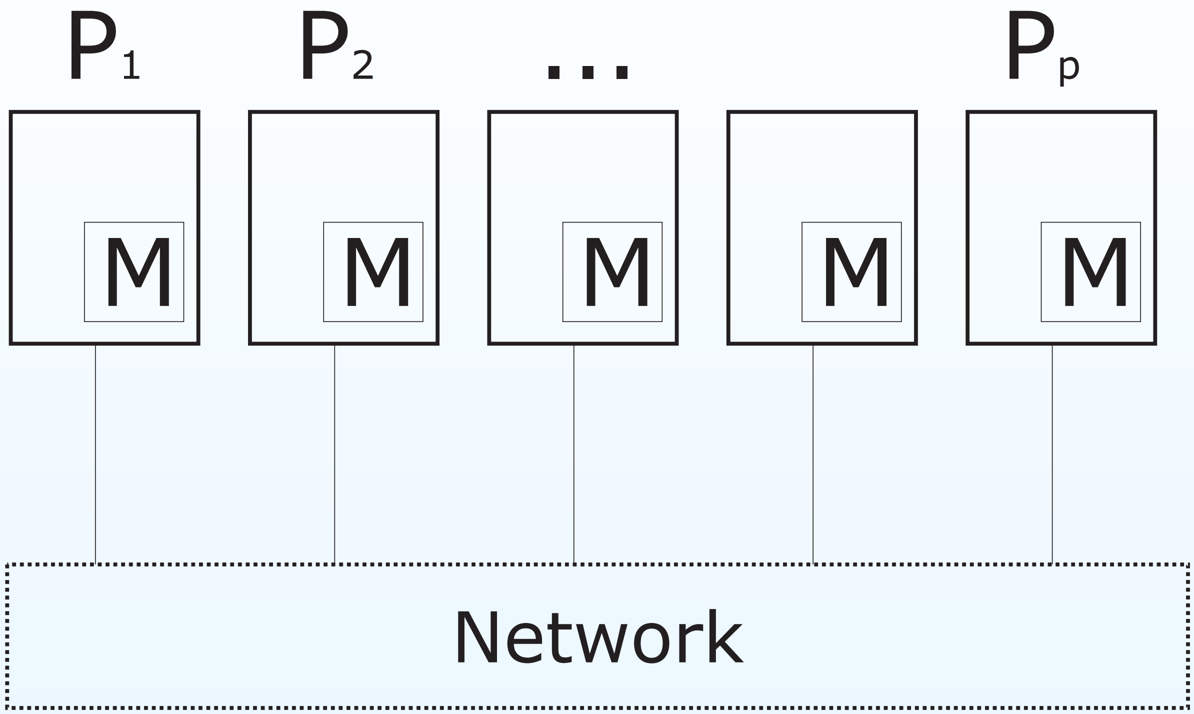
The BSP Model

The BSP model for parallel algorithms was used with some slight adaptations.

The model has been adapted here to use seconds instead of flops as a base unit for the running time:

- p identical processor/memory pairs (computing nodes), computation speed f
- Arbitrary interconnection network, latency l, bandwidth g

The BSP Model



- Programs are SPMD
- Execution takes place in *supersteps*
- Cost Formula : $T = f \cdot W + g \cdot H + l \cdot S$
- As a base unit for communications, 8-byte doubles will be used

BSP Programming

'BSP-style' programming using a conventional communications library (MPI/Cray shmem/...)

- Barrier synchronizations for creating superstep structure
- Many libraries already provide functionality for one sided communication/direct remote memory access (DRMA)

Using a specialized library (The Oxford BSP Toolset/PUB/CGMlib/...)

- Specialized communication primitives (bulk synchronous message passing/DRMA)
- Some libraries (Oxford Toolset, PUB) include optimized barrier synchronization functions and routing
- Higher level of abstraction

The BSPlib Standard

Communication primitives:

- DRMA: buffered and unbuffered `put`, `get`
- BSMP: `send` and `move`
- Synchronization
- Combining and Broadcasting

For the experiments, a BSPlib-style wrapper library was created.

BSPlib Implementations

The Oxford BSP Toolset

- Supports 3 kinds of base architecture: message passing, shared memory, DRMA
- Experiments used message passing MPI interface
- Last release from '98, compatibility issues on more modern systems

PUB

- Support for message passing and shared memory architectures
- Experiments used message passing MPI interface
- Additional support for oblivious synchronization, processor groups
- Less trouble with setup on all systems
- Advanced functionality e.g. for process migration

Other Libraries

CGMlib

- Runs on top of message passing MPI
- Includes set of algorithms for sorting, list ranking, etc.
- Abstract C++ interface
- Lists of abstract datatypes with constant size are used for data exchange

SSCRAP

- Uses MPI (message passing) or Posix (SHMEM) for data exchange
- Support for DRMA, BSMP, conventional message passing, collective operations, etc.
- 'Soft' synchronization (send or receive)
- C++ interface

'BSP-style' Programming in MPI

Approach here: a BSPlib style MPI-1 library was implemented naively (without message combining, etc.)

- Isend/Recv for data exchange
- Barrier synchronization
- Emulated DRMA on top

Advantage: no overhead for `send/put`
Drawback: high latency, presumably overhead for `get` operations

Systems used

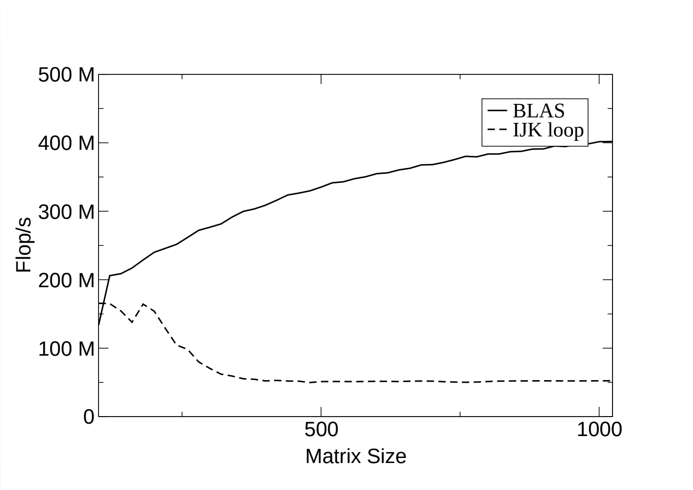
Measurements on parallel machines at the Centre for Scientific Computing:

`aracari`: IBM cluster, 64 $\times$ 2-way SMP Pentium3 1.4 GHz/128 GB of memory
(Interconnection Network: Myrinet 2000,
MPI: `mpich-gm`)

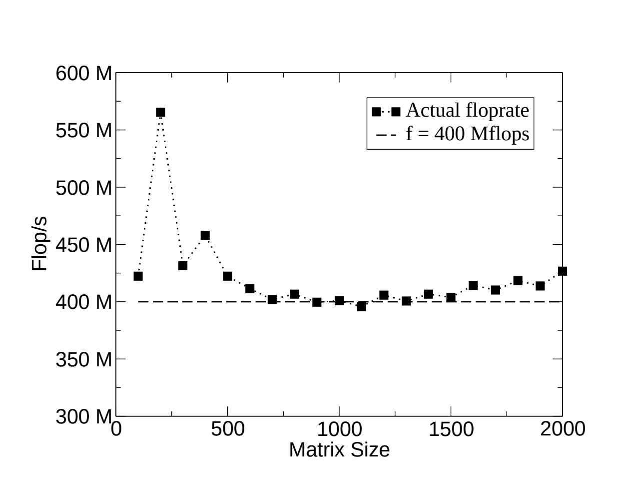
`argus`: Linux cluster, 31 $\times$ 2-way SMP Pentium4 Xeon 2.6 GHz processors/62 GB of memory
(Interconnection Network: 100Mbit Ethernet,
MPI: `mpich-p4`)

Measuring f

Measuring algorithm performance on one node:



Measuring computation time separately in one run:



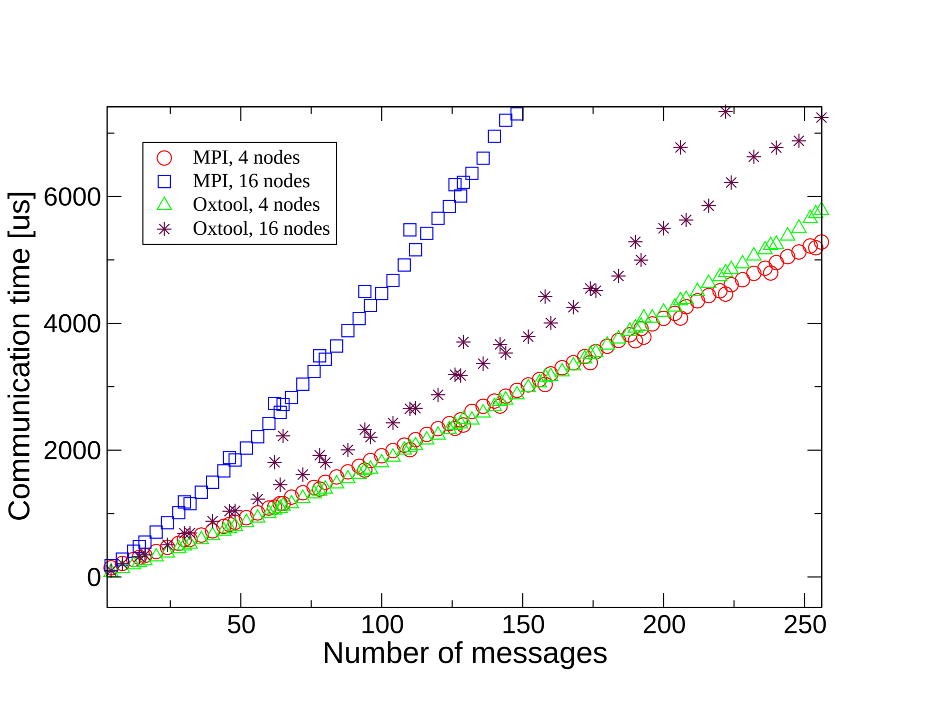
(Example for Matrix-Matrix multiplication)

Measuring g and l

Problems encountered: realistic values of g and l depend on

- The number of processors that are used
- The communications pattern
- The communication volume

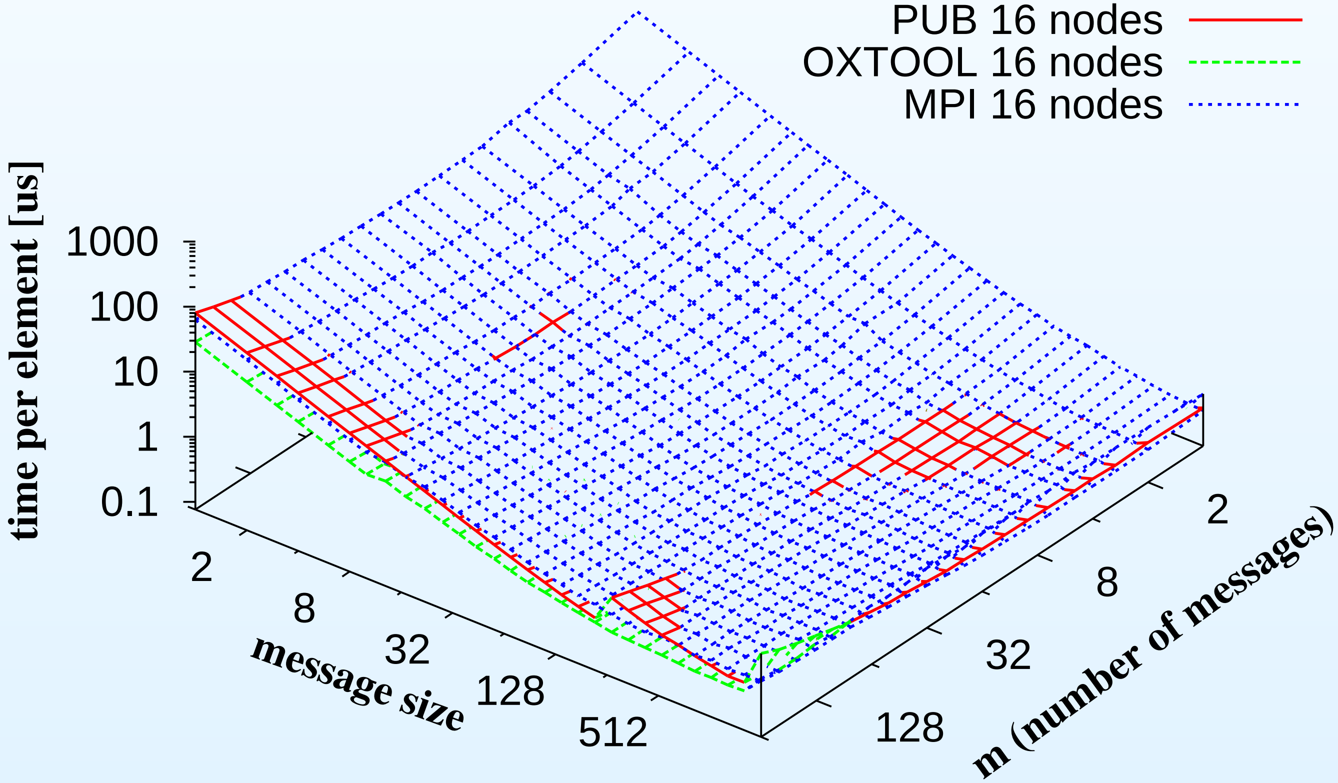
E.g. for all-to-all communication on `aracari`



Bandwidth Surface (`aracari`)

For a better picture, the effective bandwidth can be sampled depending on message size and count.

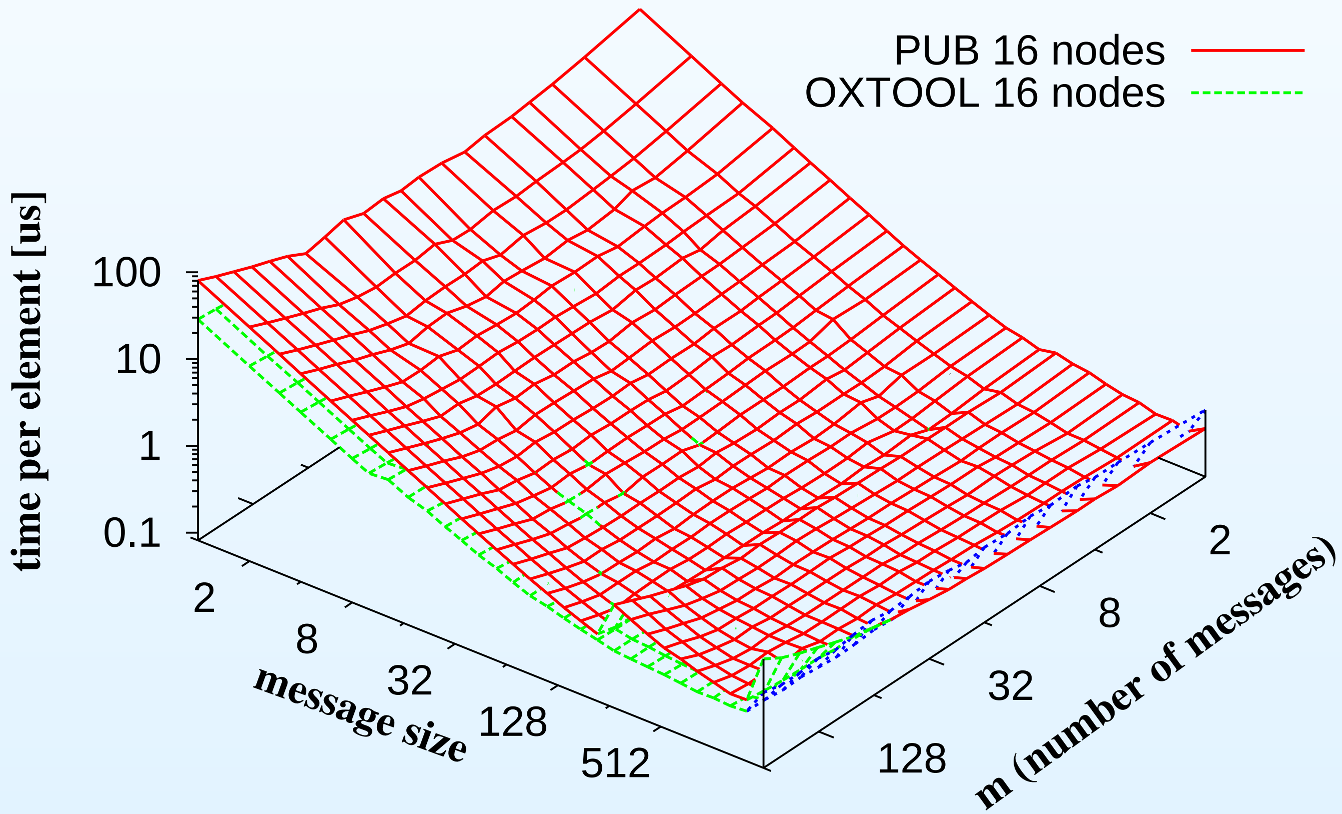
All-to-all communication on `aracari`:



Bandwidth Surface (`aracari`)

For a better picture, the effective bandwidth can be sampled depending on message size and count.

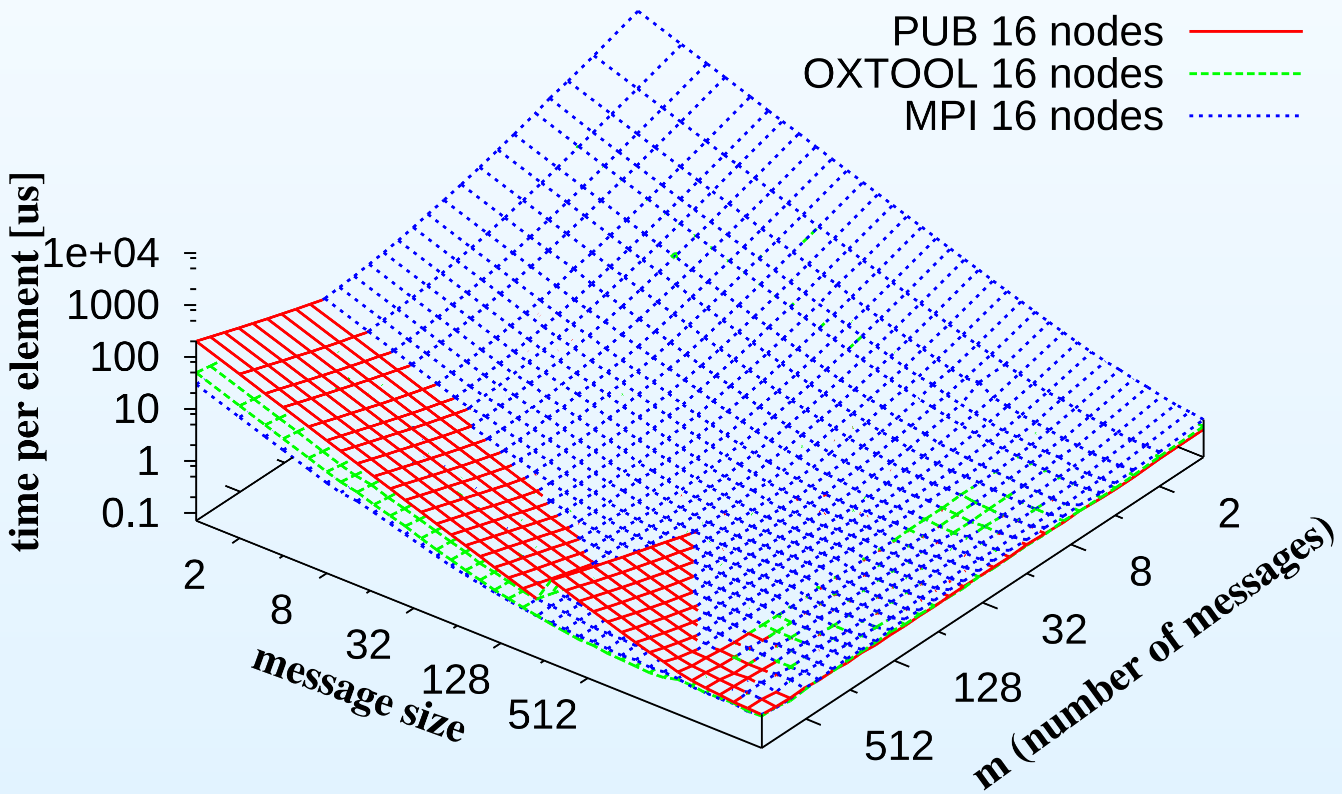
All-to-all communication on `aracari`:



Bandwidth Surface (`aracari`)

For a better picture, the effective bandwidth can be sampled depending on message size and count.

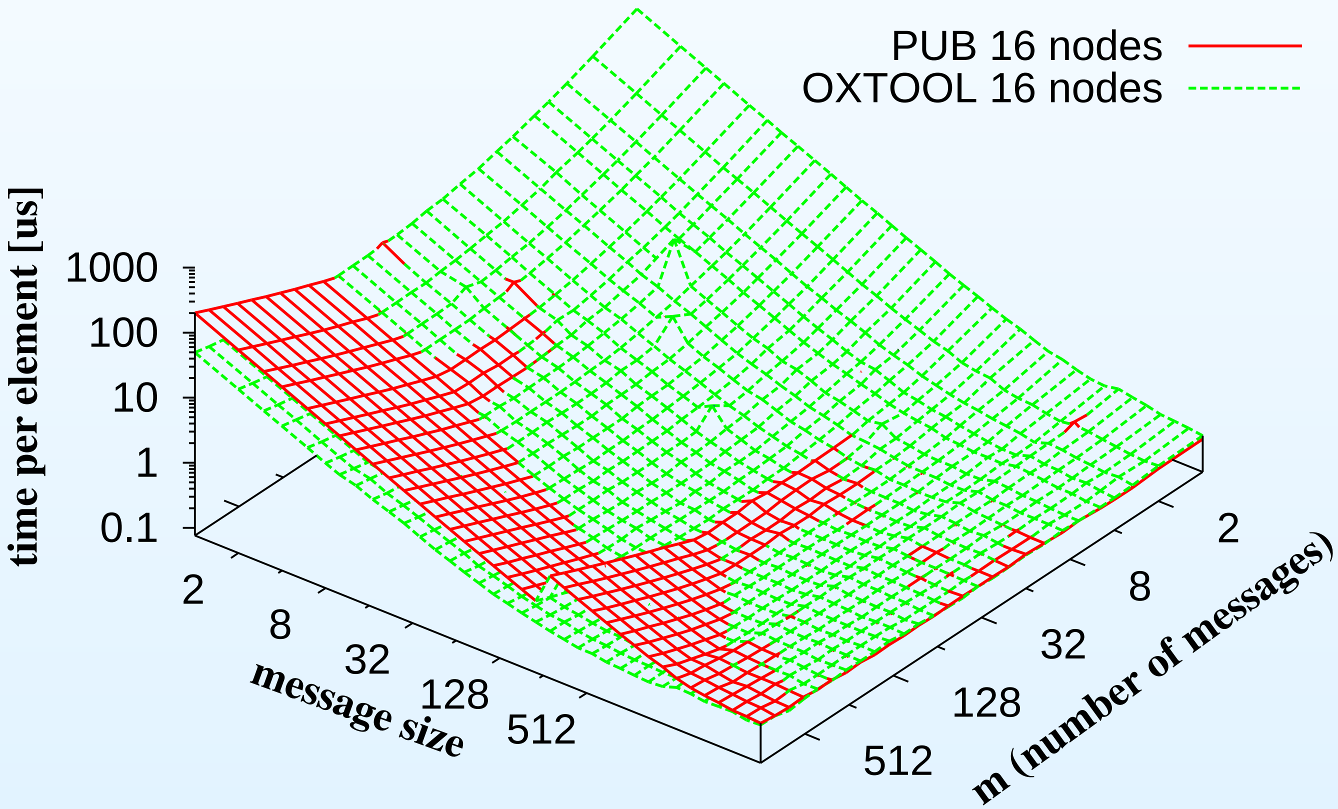
Random permutation on `aracari`:



Bandwidth Surface (`aracari`)

For a better picture, the effective bandwidth can be sampled depending on message size and count.

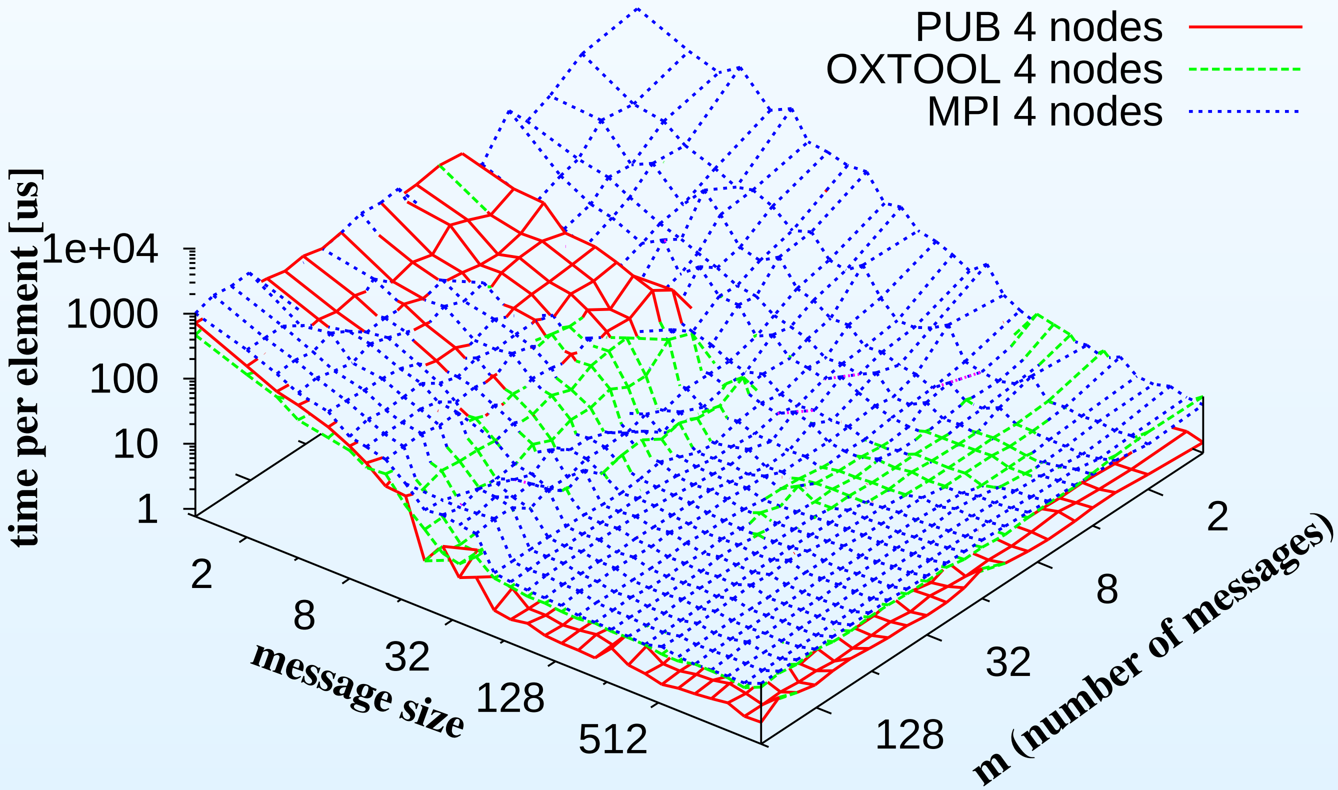
Random permutation on `aracari`:



Bandwidth Surface (`argus`)

The picture looks different on the slower communications network

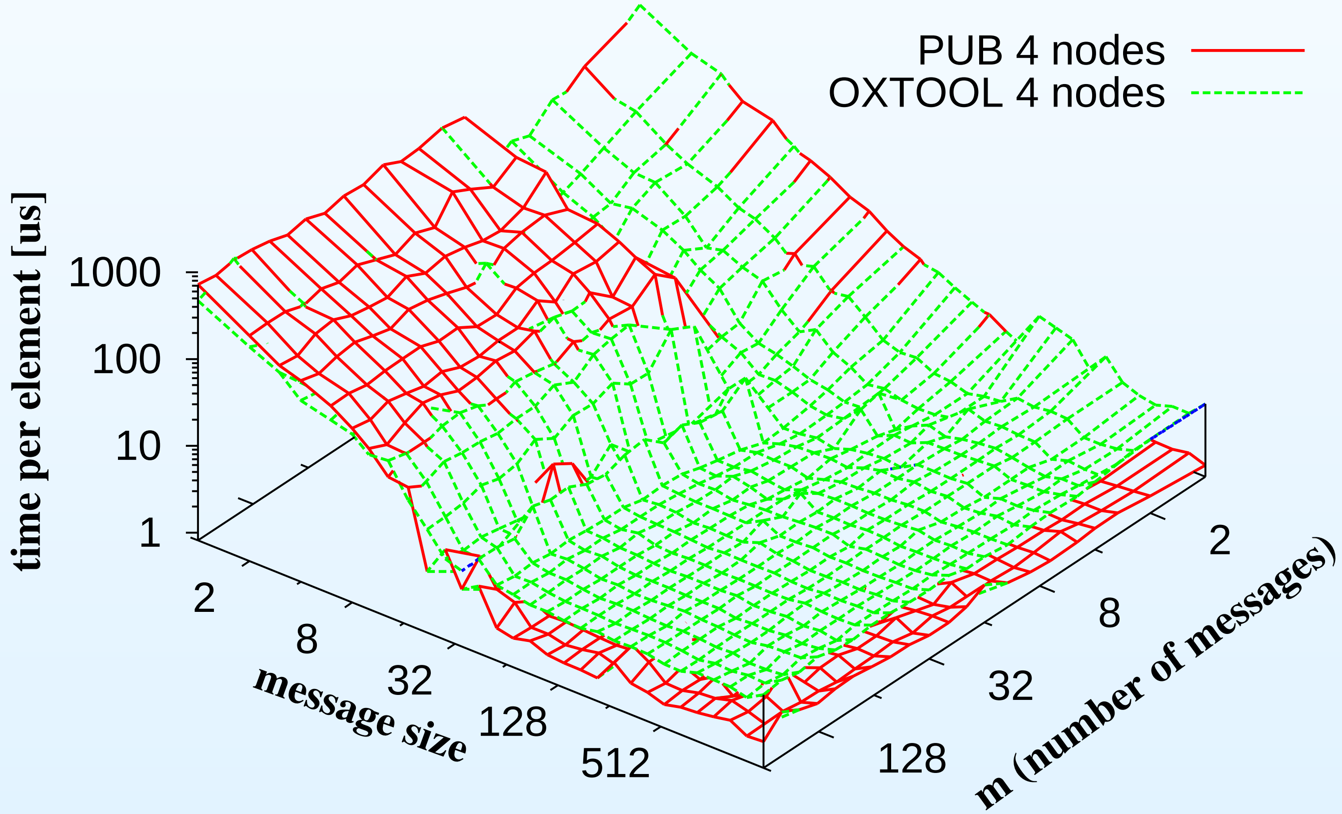
All-to-all communication on `argus`:



Bandwidth Surface (`argus`)

The picture looks different on the slower communications network

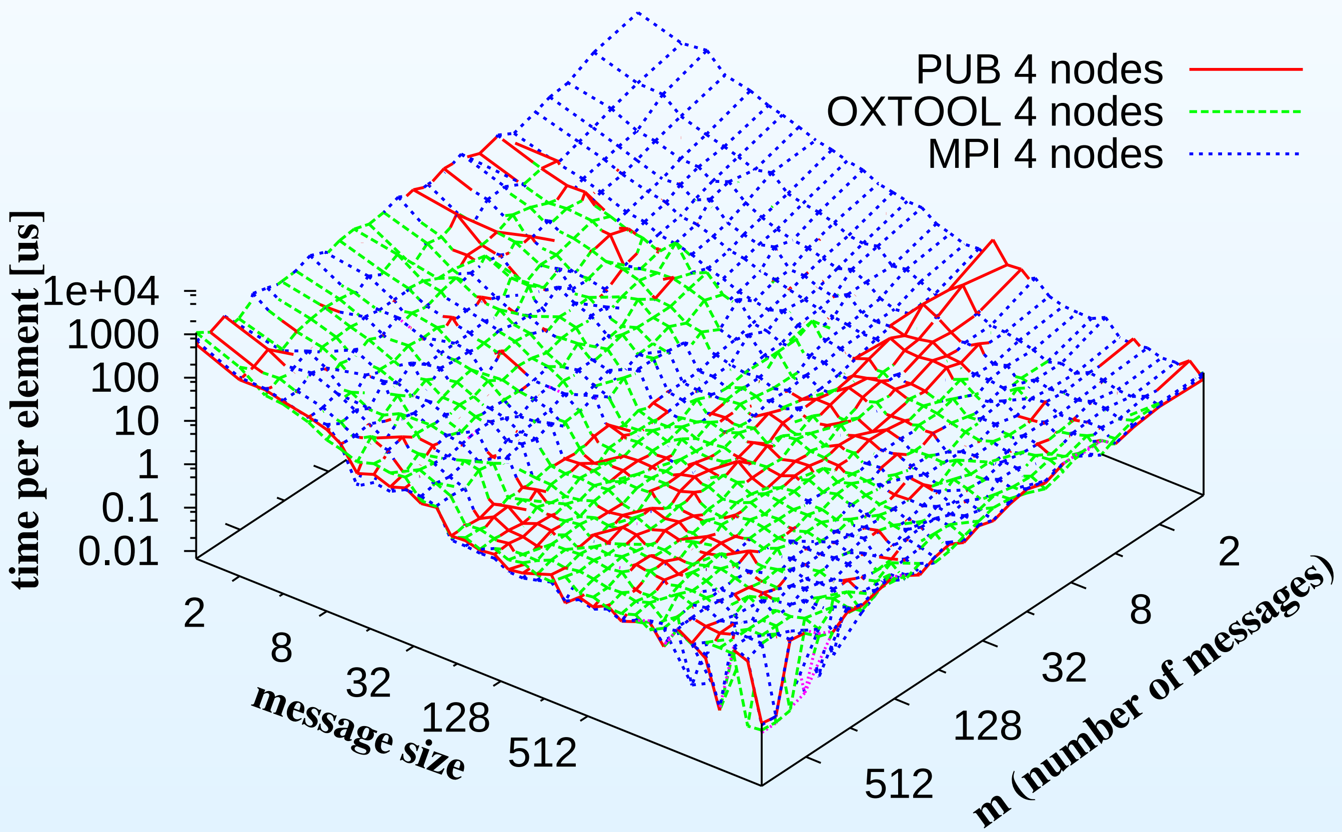
All-to-all communication on `argus`:



Bandwidth Surface (`argus`)

The picture looks different on the slower communications network

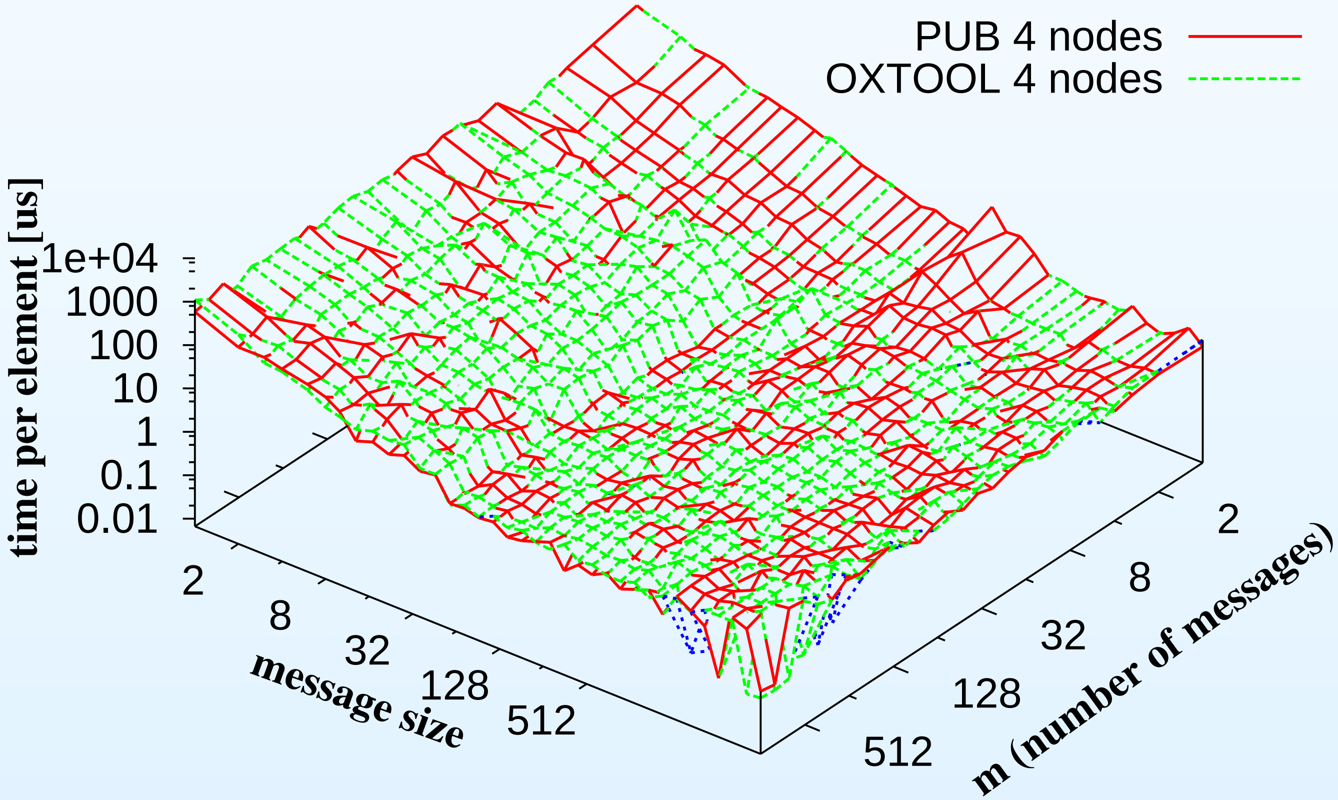
Random permutation on `argus`:



Bandwidth Surface (`argus`)

The picture looks different on the slower communications network

Random permutation on `argus`:



Latency

The latency can be measured for synchronizations preceded by different types of communication:

	MPI	**Oxtool**	**PUB**
	`aracari` *4 processors*		
l (low)	210 μs	43 μs	39 μs
l (high)	230 μs	67 μs	55 μs
l (all-to-all)	252 μs	89 μs	72 μs
	`aracari` *32 processors*		
l (low)	2203 μs	621 μs	142 μs
l (high)	2242 μs	638 μs	163 μs
l (all-to-all)	2881 μs	1250 μs	750 μs
	`argus` *4 processors*		
l (low)	5642 μs	796 μs	975 μs
l (high)	5789 μs	1442 μs	1176 μs
l (all-to-all)	5086 μs	1613 μs	871 μs

Benchmark Summary

Bandwidth depends on message count, message size and the communications pattern

On `aracari`:

- Best all-to-all performance: Oxtool
- Best random permutation performance (few messages): PUB, > 64 messages: Oxtool
- Best self communication performance (few messages): PUB, > 32 messages: Oxtool
- MPI: good performance when message size is large

Benchmark Summary

Bandwidth depends on message count, message size and the communications pattern

On `argus`:

- Best all-to-all performance: PUB
- Random permutation: little difference between PUB and Oxtool
- Best self communication performance (few messages): Oxtool
- MPI: good performance when message size and count are larger than 16/32 doubles

Benchmark Summary

Latency:

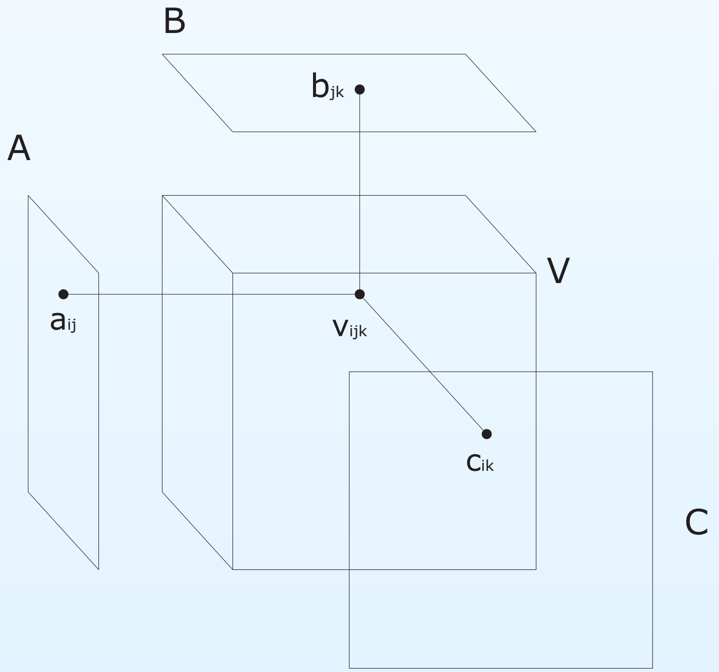
- PUB consistently has best latency (without using the faster ‘oblivious’ synchronization)
- As expected, ‘naive’ MPI library has highest latency

BSP Matrix-Matrix Multiplication

We want to compute the product of two dense $n \times n$ matrices A and B

Simple formula:

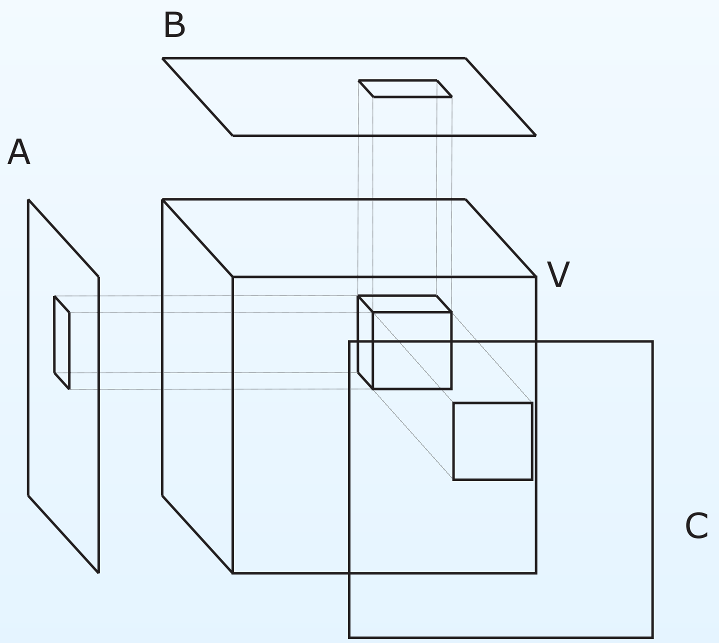
$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}, \text{ having } A = [a_{ij}], \quad B = [b_{ij}], \quad C = [c_{ij}]$$



BSP Matrix-Matrix Multiplication (2)

Block decomposition into q blocks for memory efficient parallel algorithm:

$$C_{IK} = \sum_{J=1}^{q^{1/3}} V_{IJK} \qquad \text{with } I, K = 1, 2, ..., q^{1/3}$$



Why this algorithm?

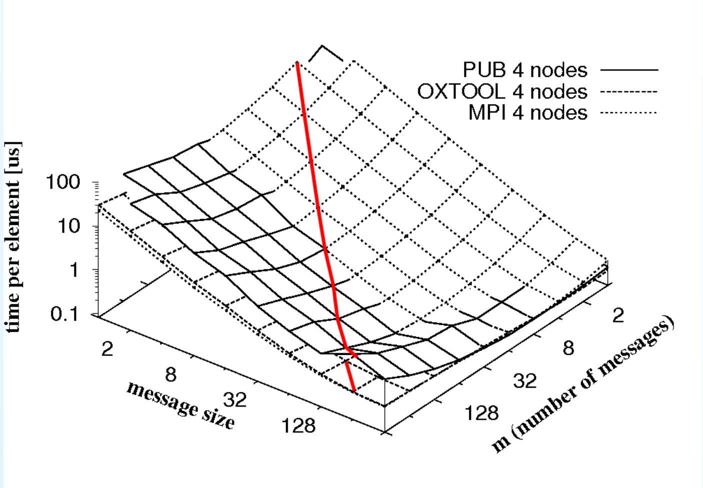
- Communication block size can be controlled by parameter q
- Message combining has to be used when using fixed initial data distribution (block-cyclic with block width $n/\sqrt{p}$)
- Can be compared e.g. to PBLAS
- ‘Nice’ version can predistribute the blocks before the computation to avoid spikes because of data distribution

Prediction model

BSP running time:

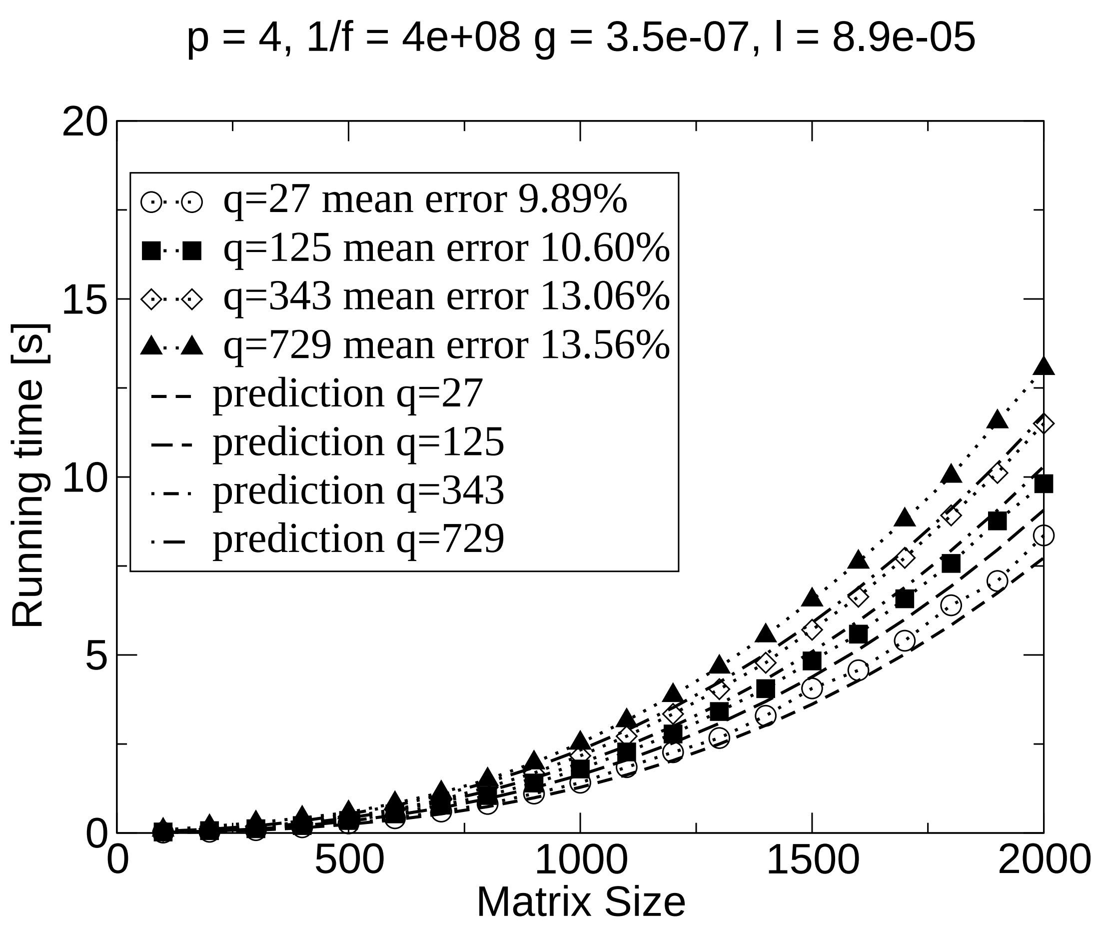
$$\begin{aligned} T \quad = \quad & f \cdot \left\lceil \frac{q}{p} \right\rceil \cdot \frac{n^3}{q} + \\ & g \cdot \left\lceil \frac{q}{p} \right\rceil \cdot \frac{n^2}{q^{2/3}} \cdot \left(2 + 1/q^{1/3}\right) + \\ & l \cdot 2 \left\lceil \frac{q}{p} \right\rceil \end{aligned}$$

Two matrices are transferred row by row
$\rightarrow$ value of g is taken from the red line as value for maximum matrix size



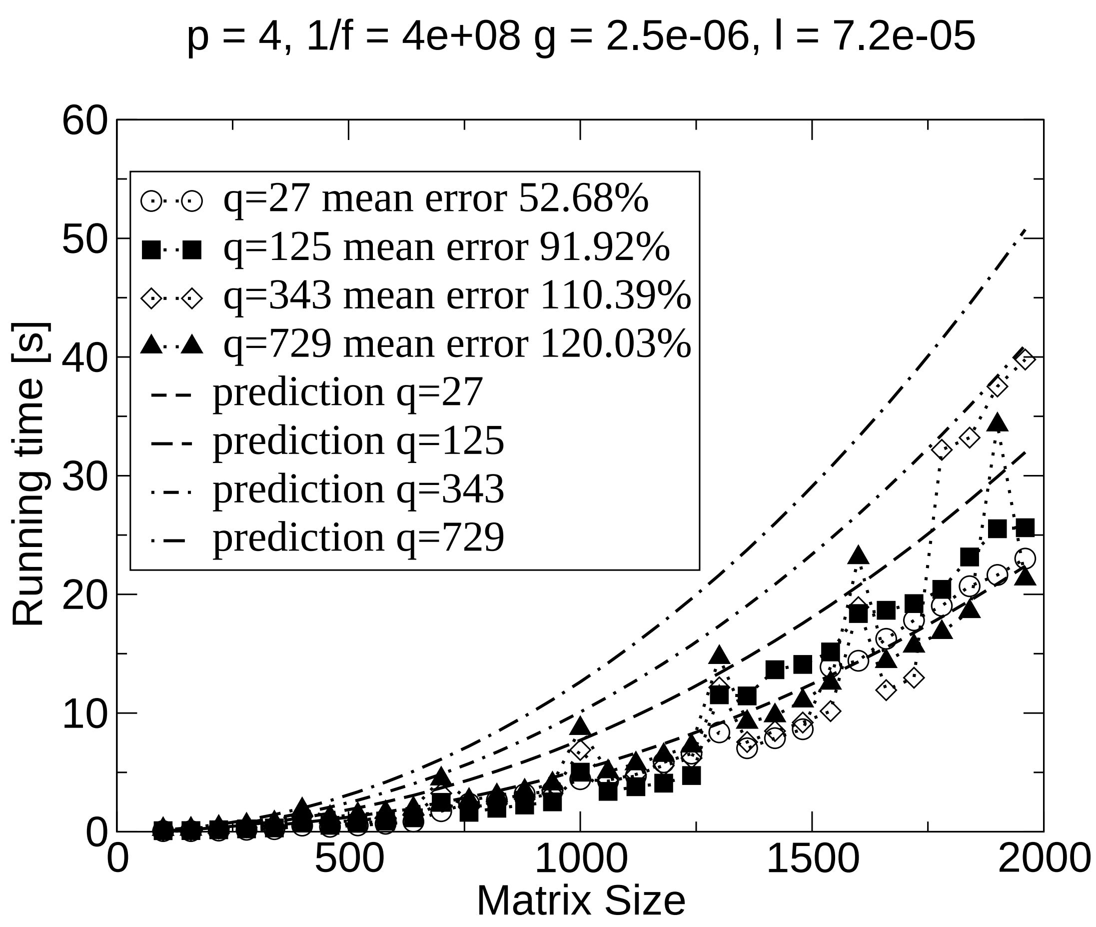
Prediction results on `aracari`

Oxtool, using 4 processors



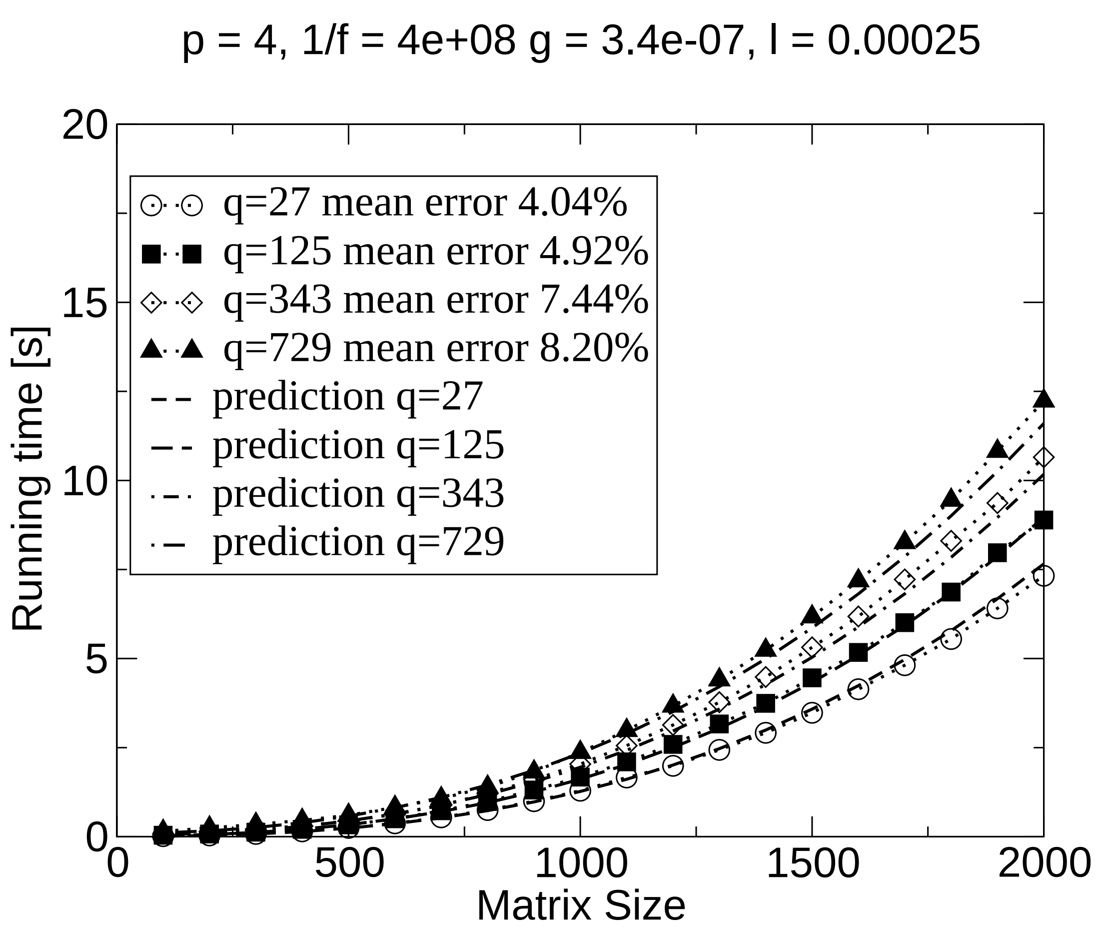
Prediction results on `aracari`

PUB, using 4 processors



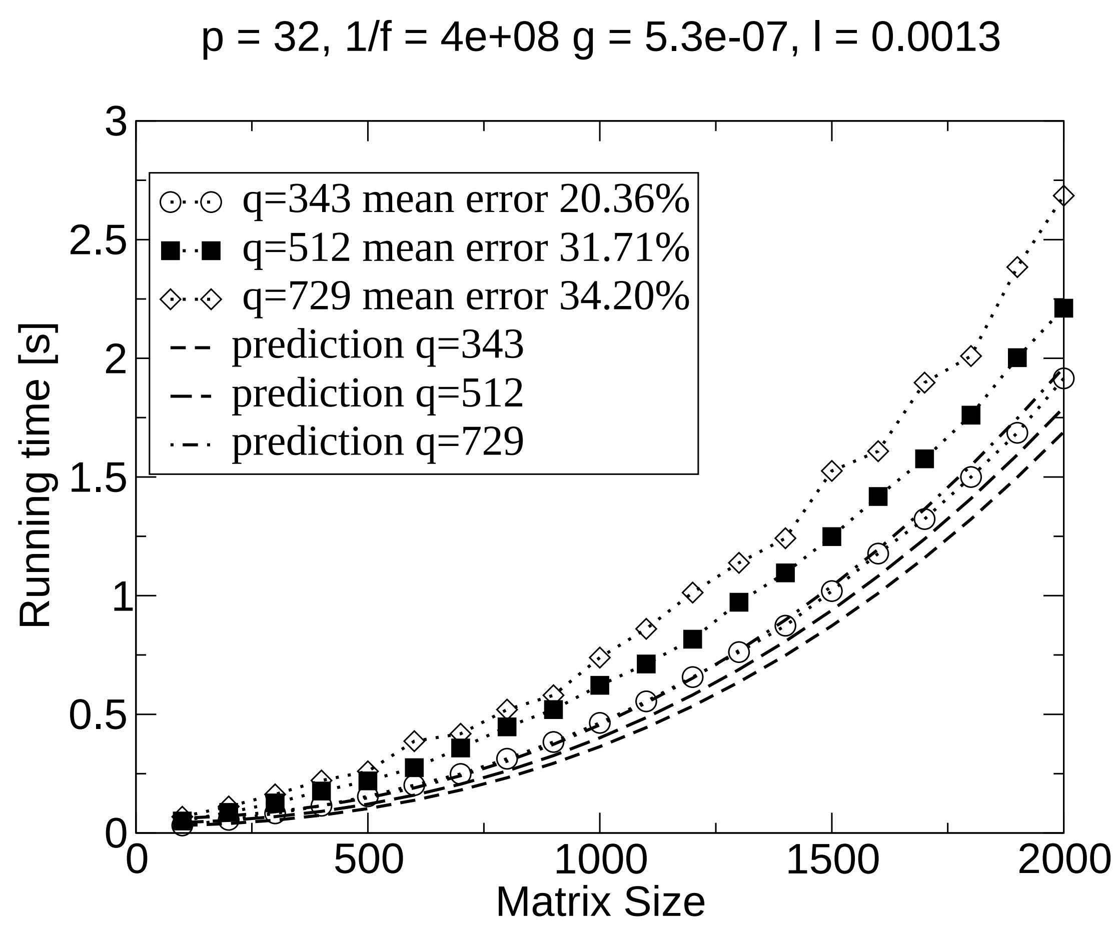
Prediction results on `aracari`

MPI, using 4 processors



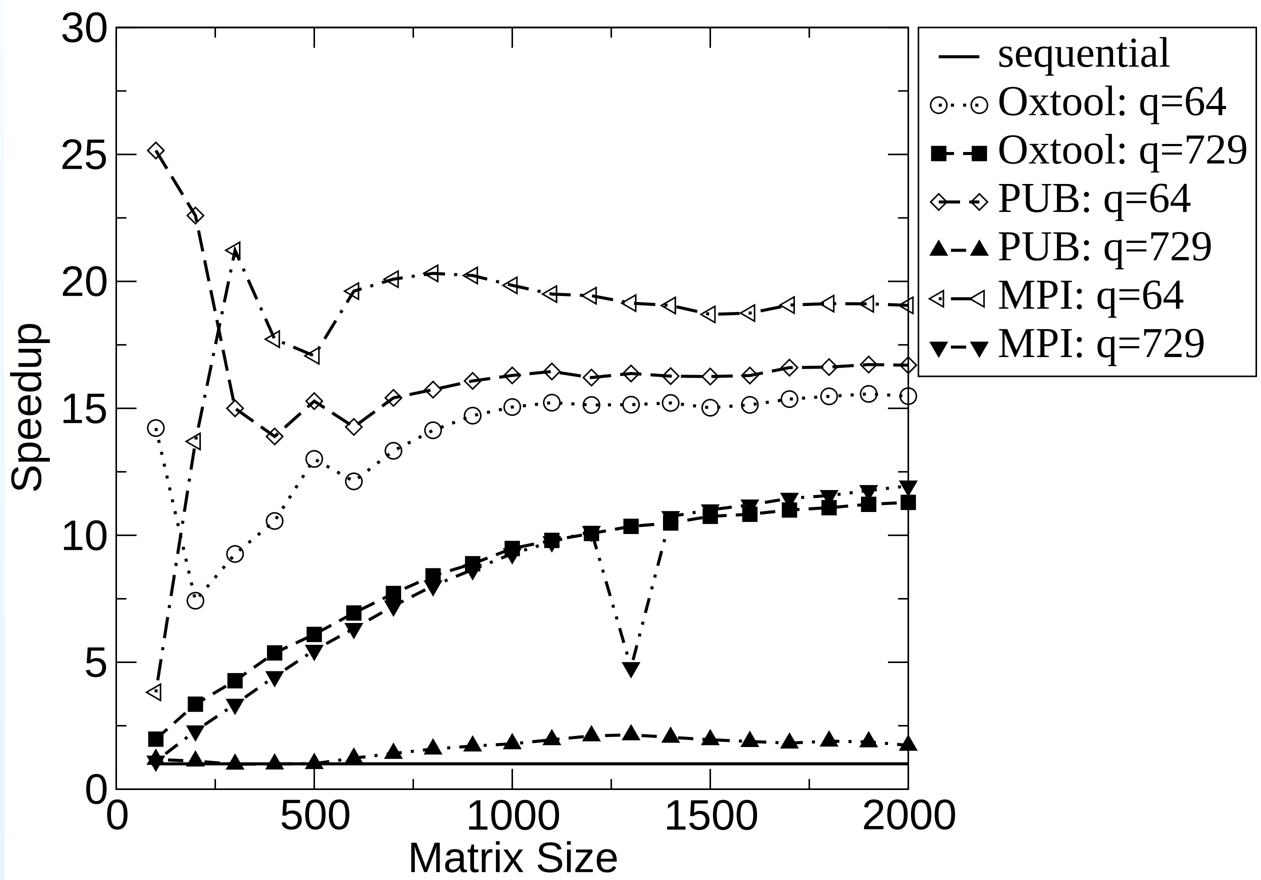
Prediction results, more processors

Oxtool, using 32 processors



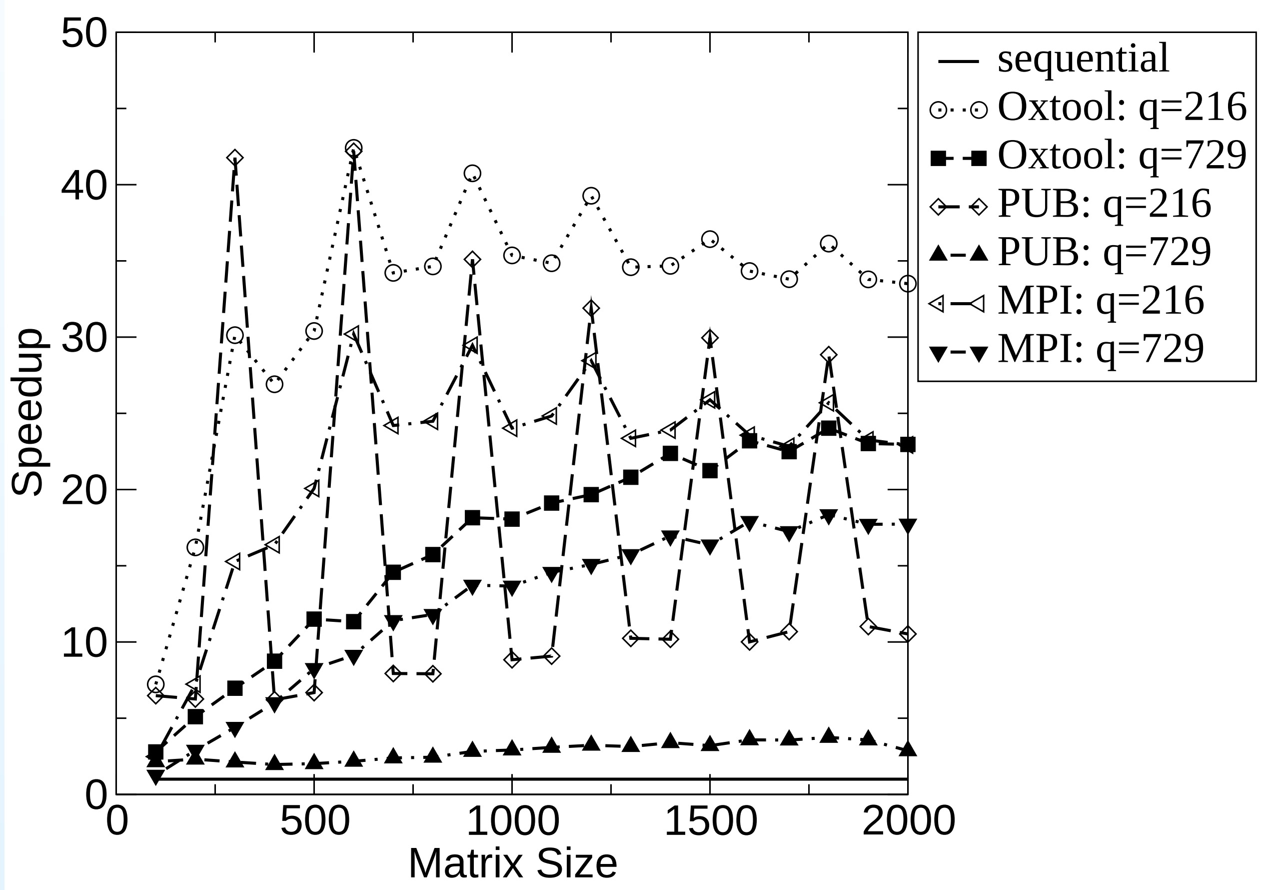
Speedup Results (1)

aracari, using 16 processors

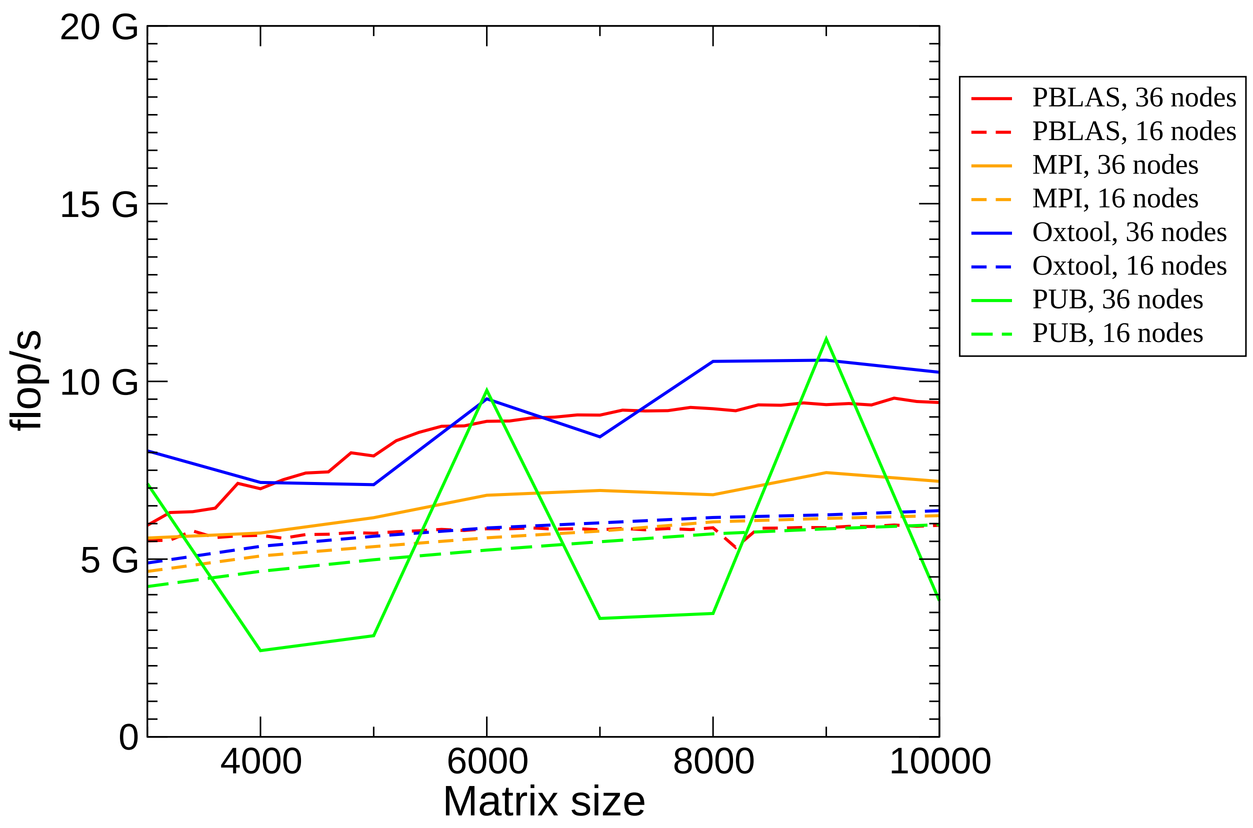


Speedup Results (2)

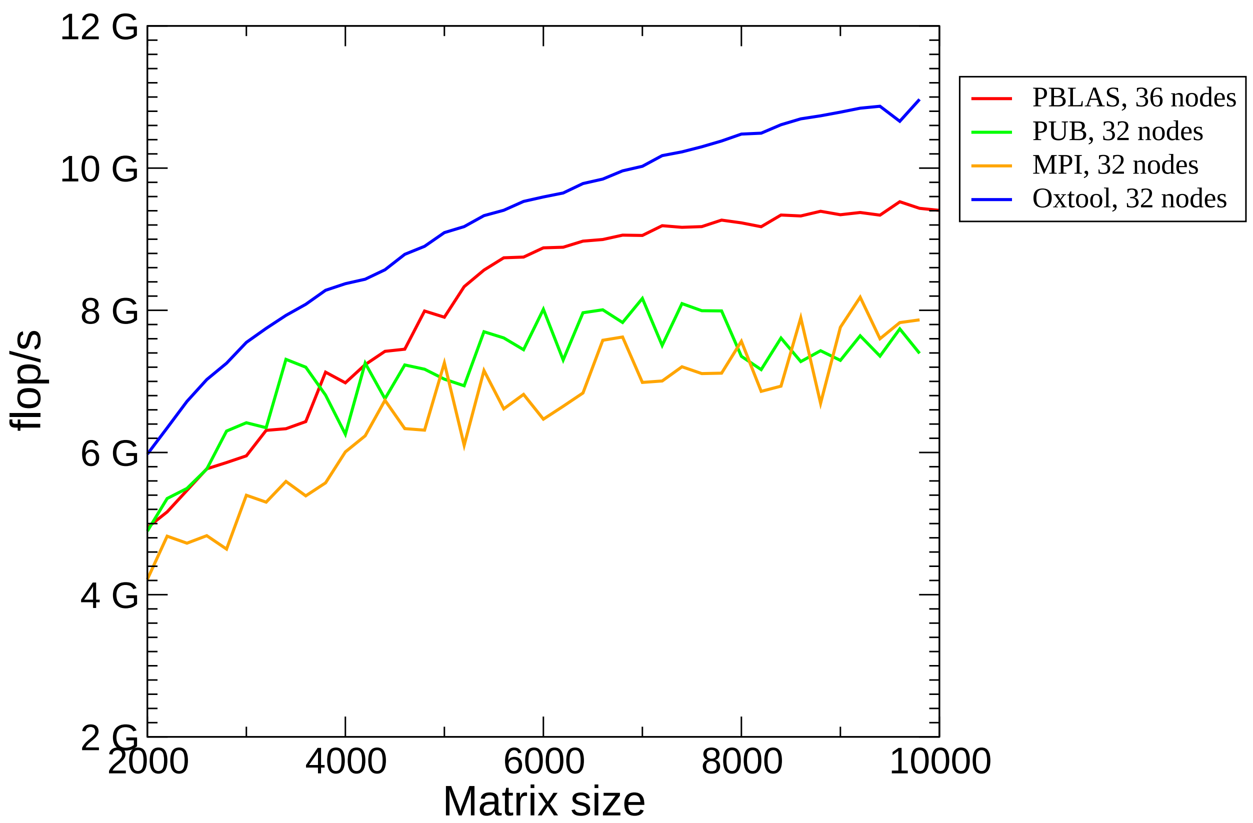
`aracari`, using 32 processors (spikes when matrix size `mod` $6 == 0$)



Performance Comparison with PBLAS



Performance Comparison with PBLAS



Summary

- Despite restrictions due to BSP model, all implementations reach good speedup on `aracari` when number of blocks is low
- Overall benchmark results look better for PUB
- Oxtool has best matrix multiplication performance on `aracari` (Myrinet)
- PUB has best matrix multiplication performance on `argus` (Ethernet)
- Predictably no real speedup on `argus`, due to slow communications network and fast nodes
- Performance of simple BSP algorithm is comparable with PBLAS

Further Work

- Run experiments on shared memory machine
- Use more different communication patterns for benchmarking
- Study other algorithms with different communication patterns
- Keeping simplicity, extend prediction model for more accuracy