

MPI.NET

High-Performance Message Passing in C# and .NET

Douglas Gregor and Andrew Lumsdaine
Open Systems Lab @ Indiana University
{dgregor, lums}@osl.iu.edu



MPI in C#

- MPI is a multi-language API standard
- Higher-level languages pose problems:
 - Virtual machine, garbage collector overhead
 - Prevalence of user-defined types
 - Typically forces a trade-off between usability and performance
- C# appears to enable clean, usable interfaces with near-native performance



Gathering Hostnames

- A simple task with MPI:
 - Have each process query its host name
 - Gather the host names to the root node
 - Sort and print the host names

- Example output:

thumb001

thumb002

thumb003

thumb004



```
int size, rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

char name[MPI_MAX_PROCESSOR_NAME];
int resultlen;
MPI_Get_processor_name(name, &resultlen);

int *rbuf = (int*)malloc(sizeof(int) * size);
MPI_Gather(&resultlen, 1, MPI_INT, rbuf, 1, MPI_INT, 0, MPI_COMM_WORLD);

int *rcounts = (int*)malloc(sizeof(int) * size);
int *rdispls = (int*)malloc(sizeof(int) * size);
int cnt = 0;
for (int i = 0; i < size; i++) {
    rcounts[i] = rbuf[i]+1;
    cnt += rcounts[i]+1;
    if (i) rdispls[i] = rdispls[i-1]+rbuf[i]+1;
    else rdispls[i] = 0;
}
char *rnamebuf = (char*)malloc(cnt * sizeof(char));
MPI_Gatherv(name, resultlen, MPI_CHAR, rnamebuf, rcounts, rdispls, MPI_CHAR, 0,
    MPI_COMM_WORLD);
if (rank == 0) {
    char **hostnames = (char**)malloc(size * sizeof(char*));
    for (int i = 0; i < size; ++i)
        hostnames[i] = rnamebuf + rdispls[i];
    qsort (hostnames, size, sizeof(char*), strcmp);
    for (int i = 0; i < size; ++i)
        printf("%s\n", hostnames[i]);
}
```

Gathering Hostnames – C#

```
Communicator comm = Communicator.world;
string[] hostnames =
    comm.Gather(MPI.Environment.ProcessorName, 0);
if (comm.Rank == 0) {
    Array.Sort(hostnames);
    foreach(string host in hostnames)
        Console.WriteLine(host);
}
```

- Important features for C#:
 - Simple MPI interfaces
 - Transparent support for all types



Generic Point-to-Point Send

```
public class Communicator {  
    public void Send<T>(T value, int dest, int tag) {  
        BinaryFormatter format = new BinaryFormatter();  
        using (MemoryStream stream = new MemoryStream()) {  
            format.Serialize(stream, value);  
            unsafe {  
                fixed (byte* buffer = stream.GetBuffer())  
                Unsafe.MPI_Send(new IntPtr(buffer),  
                    (int)stream.Length, Unsafe.MPI_BYTE, dest,  
                    tag, comm);  
            }  
        }  
    }  
}
```

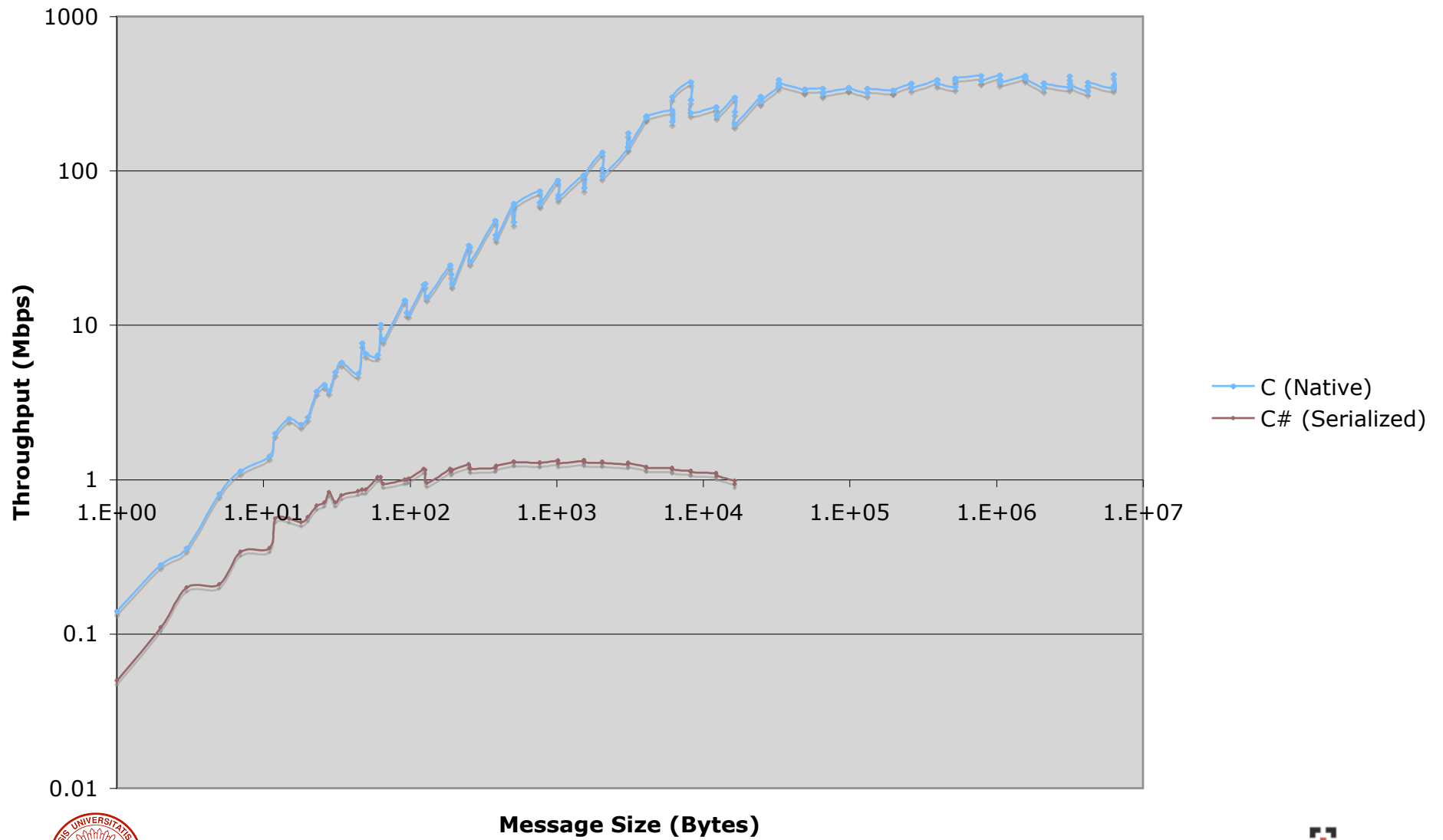
Serialize (points to `format.Serialize(stream, value);`)

Pin Memory (points to `fixed (byte* buffer = stream.GetBuffer())`)

Call Native MPI (points to `Unsafe.MPI_Send(...)`)



NetPIPE Performance



Sending Integers, Fast

- MPI is good at transmitting primitive types

```
public void SendInt(int value, int dest, int tag)
{
    unsafe {
        fixed (int* valuePtr = &value) {
            Unsafe.MPI_Send(new IntPtr(valuePtr), 1,
                Unsafe.MPI_INT, dest, tag, comm);
        }
    }
}
```



Flexible, Efficient MPI Send

```
public void Send<T>(T value, int dest, int tag) {
    MPI_Datatype datatype = DatatypeCache<T>.datatype;
    if (datatype != MPI_DATATYPE_NULL) {
        // Fast path: send directly with MPI
        unsafe {
            fixed (void* valuePtr = &value)
                Unsafe.MPI_Send(new IntPtr(valuePtr), 1, datatype,
                                dest, tag, comm)
        }
    } else {
        // Slow path: serialize and send
    }
}
```

Error: can't take address

C#	MPI
short	MPI_SHORT
int	MPI_INT
float	MPI_FLOAT
double	MPI_DOUBLE



Flexible, Efficient MPI Send

```
public void Send<T>(T value, int dest, int tag) {
    MPI_Datatype datatype = DatatypeCache<T>.datatype;
    if (datatype != MPI_DATATYPE_NULL) {
        // Fast path: send directly with MPI
        unsafe {
            IntPtr valuePtr = Subvert.LoadAddress(ref value);
            Unsafe.MPI_Send(valuePtr, 1, datatype,
                dest, tag, comm)
        }
    } else {
        // Slow path: serialize and send
    }
}
```

C#	MPI
short	MPI_SHORT
int	MPI_INT
float	MPI_FLOAT
double	MPI_DOUBLE

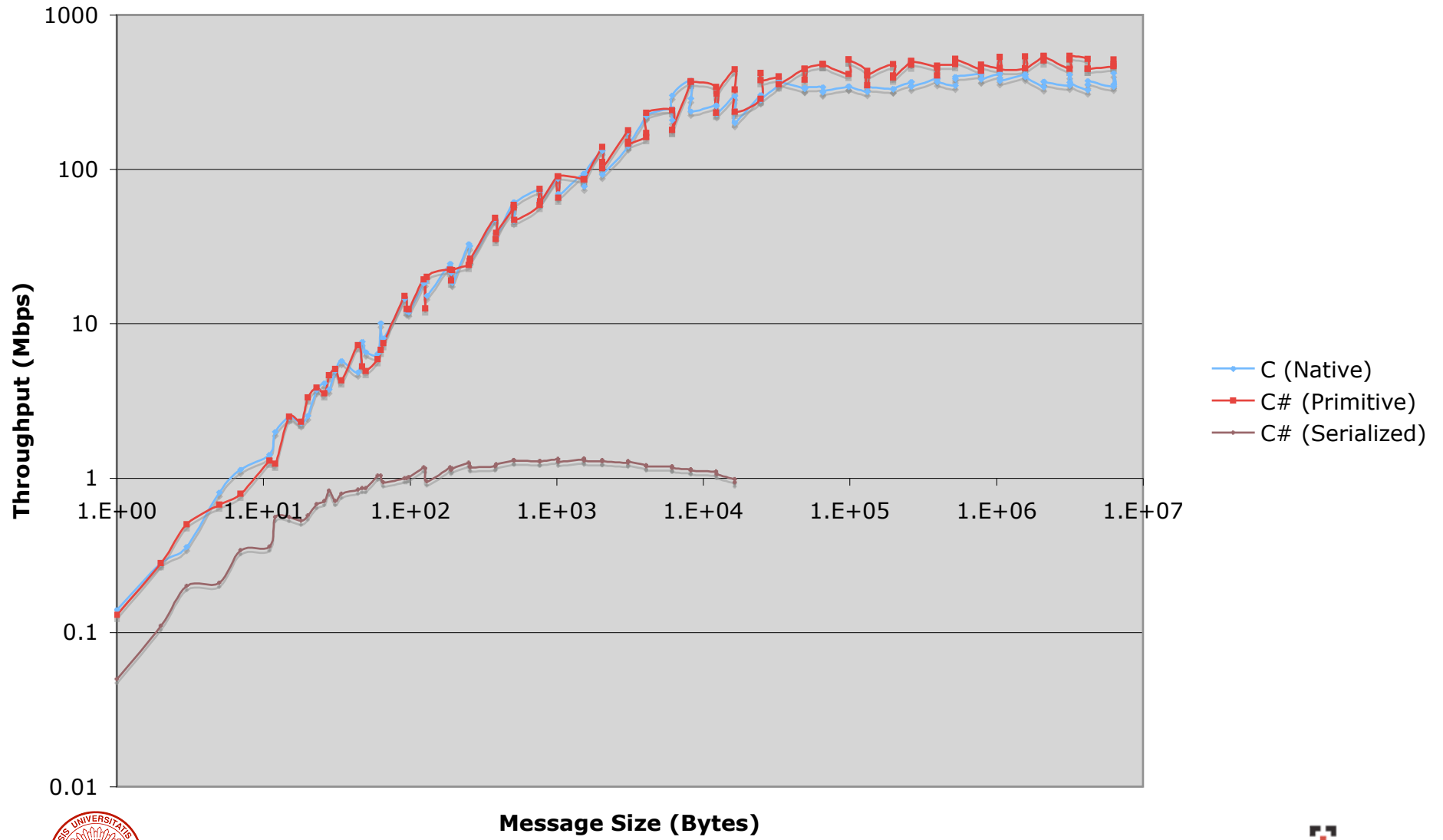


Escaping to CIL Assembler

```
.class public auto Subvert extends [mscorlib]System.Object
{
    .method public hidebysig static
        native int LoadAddress<T>(!!T& 'value') cil managed
    {
        // Code size          14 (0xe)
        .maxstack 1
        .locals init ([0] !!T& pinned ptr)
        ldarg.0
        stloc.0
        ldloc.0
        conv.i
        newobj instance void [mscorlib]System.IntPtr::.ctor(void*)
        ret
    }
}
```



NetPIPE Performance



Value Types as Derived Datatypes

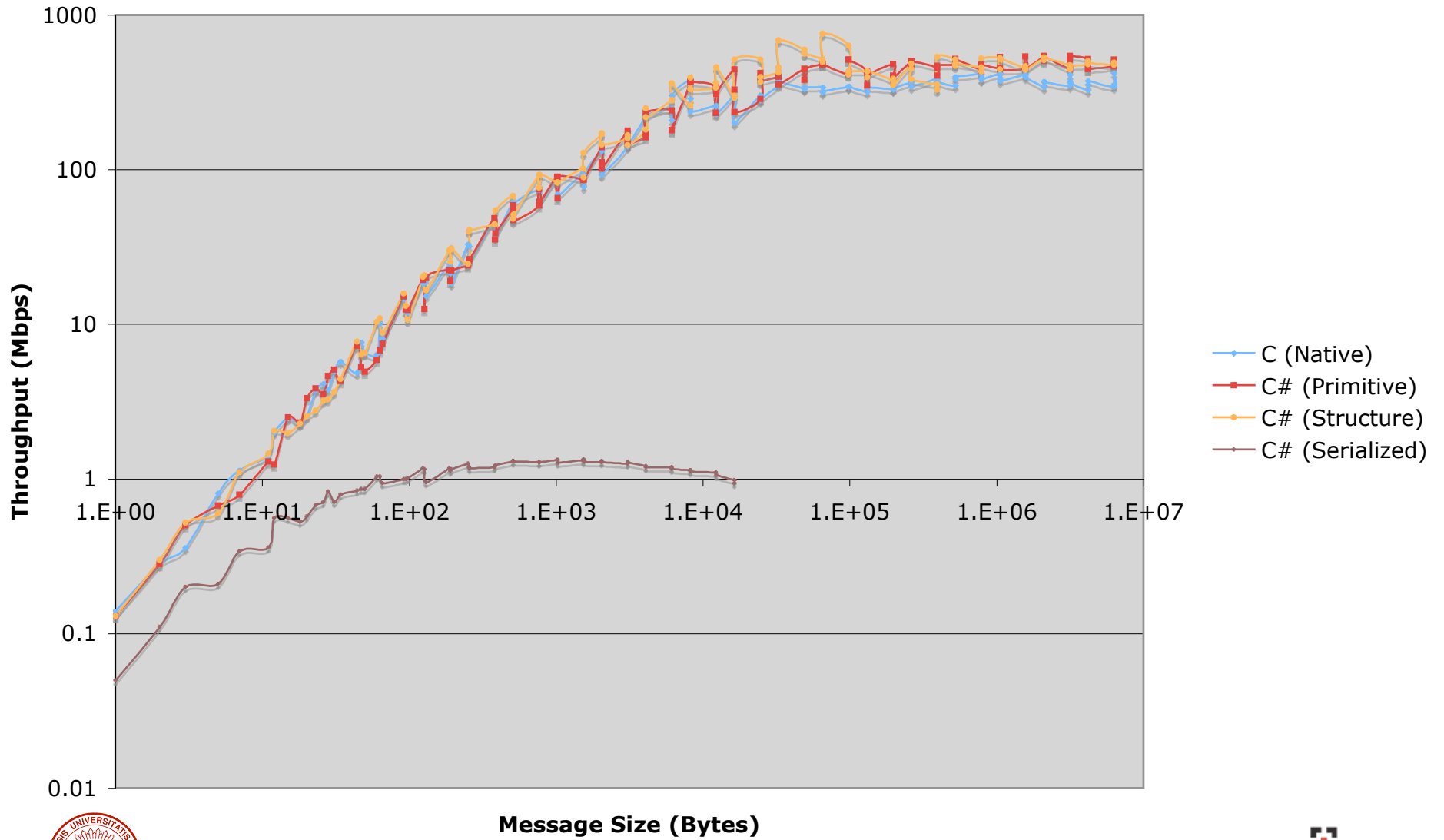
```
public struct Point {  
    public float x;  
    public float y;  
    public float z;  
}
```

```
public class Point {  
    public float x;  
    public float y;  
    public float z;  
}
```

- Automatically generate MPI derived datatypes from C# value types
 - Run-time reflection provides types/offsets
 - Optimization is transparent to the user



NetPIPE Performance



Non-Blocking Communication

- Similar to blocking communication, but...
 - Need to pin memory until request completes
 - Value types need to be received on the heap
- Memory pinning with GCHandle:

```
GCHandle handle =  
    GCHandle.Alloc(value, GCHandleType.Pinned);  
  
// when request completes...  
handle.Free();
```



MPI Collectives

```
string myString = computeLocalString();  
string concat = comm.Allreduce(myString, (x, y)=>x+y);
```

- ❑ Collectives are algorithms involving all processes in a communicator
- ❑ Same design approach:
 - Complete support for user-defined types
 - Use *C# delegates* to express reduction operations
- ❑ Requires new implementation in C#



Optimizing `Allreduce`

□ Primitive types, predefined operations

```
comm.Allreduce(myInt, Operation<int>.Plus);
```

■ Map to native call

```
MPI_Allreduce(..., MPI_INT, MPI_SUM...)
```

□ Value types, user-defined operations

```
comm.Allreduce(myPoint, PointSum);
```

■ `MPI_Op_create` to build “thunk” into VM

■ Map to native call to `MPI_Allreduce`



MPI.NET Summary

- Design & optimization methodology for elegant interfaces to native libraries
- MPI.NET is available as open source
 - Covers ~95% of MPI 1.1
 - BSD-like license
 - Support for multiple platforms:
 - Windows: Microsoft .NET with MS-MPI
 - Unix: Mono with Open MPI, LAM/MPI, MPICH2



<http://www.osl.iu.edu/research/mpi.net/>

