Using C# for high performance network programming

Low latency
High throughput
(10-100Gbps)

Jonny Shipton
CL, University of Cambridge
What is a NAT?

Alice

Bob

Cathy

NAT

www.google.co.uk

www.bbc.co.uk

212.58.244.26:80

Reality
What is a NAT?

Client’s POV

Alice

Bob

Cathy

www.google.co.uk

www.bbc.co.uk
What is a NAT?

Server’s POV

Nathan

www.google.co.uk

www.bbc.co.uk
## Why C#?

<table>
<thead>
<tr>
<th></th>
<th>Verilog</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>🟢🟢🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Scalable to 100Gbps+</td>
<td>🟢🟢🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Skill base</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td>Comfy language features</td>
<td>😞</td>
<td>🟢🟢🟢</td>
</tr>
</tbody>
</table>
Why C#?

Methods

```
try
{
  SetSendIntf(0x01);

  var values = from uint i in Enumerable.Range(0, 10)
  join ulong v in dataplane.tdata
  on 1 equals v & 0xFF
  where (v & 0xFF00) == 0x1000
  select v.Extract_Bytes(until_byte: i);

  foreach (var v in values.Cast<int>())
  {
    var task = new Task<TCPWrapper>(() => new TCPWrapper(dataplane.tdata, v, 10));
    task.Start();
    task.ContinueWith(tcp => TCP(null, null, tcp.Result, null)).Wait();
  }
}
```

LINQ
Generics
OOP
Concurrency

Loops

Exceptions

Memory Management

Also iterators, properties, inheritance, libraries, dynamic variables, safe expressive type system, …

Lambdas
Why C#?

C# vs. Verilog simulation

Hardware Time (mins) to compile and run

Time (mins) to compile and run

C# vs. Verilog simulation
Compile C# to Verilog?!

Sequential/Synchronous Semantics

C# → CIL

Concurrent Semantics

F# → CIL

Kiwi

CIL → Verilog

Synthesis

Verilog → Hardware
else if (eth.Ethertype == EthernetWrapper.EtherType.ARP) {
        && arp.HardwareType == ARPWrapper.HwType.Ethernet
        && arp.ProtocolType == (ushort)EthernetWrapper.EtherType.IPv4
        && arp.HardwareAddressLength == 6
        && arp.ProtocolAddressLength == 4)
    {
        Kiwi.Pause(); // Reduces Kiwi compile time
        if (arp.TargetProtocolAddress == myInsideIP || arp.TargetProtocolAddress == myOutsideIP) // Request for a MAC I own
        {
            uint intfIp = arp.TargetProtocolAddress;
            ulong intfMac = macOfIntf(dataplane.GetReceiveIntf());

            // Update arp packet to use as response
            arp.TargetHardwareAddress = arp.SenderHardwareAddress;
            arp.TargetProtocolAddress = arp.SenderProtocolAddress;
            arp.SenderHardwareAddress = intfMac;
            arp.SenderProtocolAddress = intfIp;

            // Update ethernet fields
            eth.DestinationMac = arp.TargetHardwareAddress;
            eth.SourceMac = intfMac;

            Kiwi.Pause();

            // Send reply
            SetSendIntf(dataplane.GetReceiveIntf());
            SendFrame(pkt_size);
        }
    }
}
Look at the code

```csharp
if (arp.Operation == ARPWrapper.Op.Request ...) {
}

public Op Operation {
    get { return (Op)data.Get16(Offset + 6); }
    set {
        data.Set16(Offset + 6, (ushort)value);
    }
}
```
Limitations

• No truly dynamic allocation (yet)
  • Array sizes must be determined statically
  • Objects can be allocated and used within loops

• Library support less mature

• Complexity -> longer compile times

• No reflection, dynamic invoke, etc.

• Clock cycles are more precious, so we want to do more in each one
  • Asynchronous – Kiwi flattens code
Thanks to:

• Nik Sultana
• Salvator Galea
• David Greaves
• Networks-as-a-Service (Naas-project.org)
• EPSRC
• Kynesim

Kiwi

Google “Kiwi compiler”
http://www.cl.cam.ac.uk/~djh11/kiwi/
http://www.cl.cam.ac.uk/research/srg/hans/hprls/orangepath/kiwi.html

NetFPGA-SUME
http://netfpga.org/site/#/systems/1netfpga-sume/details/