Quants Coding CUDA in .NET: Pitfalls and Solutions

Ben Eimer
GPU Technology Conference – San Jose, CA
April 6, 2016
Outline

• What is the problem we are trying to solve?

• The Hybrid Object Linear Algebra library (HOLA)

• Pitfalls/Solution

• Performance Data
Advisory services, technology solutions, leaders in debt & derivatives

Chatham Financial is a recognized leader in financial risk management specializing in foreign currency, interest rate and commodity risk. Our hedge accounting, valuations and derivatives and debt solutions are retained by clients worldwide.

1,800+ Clients
Both public and private, rely on Chatham for end-to-end hedge advisory, execution, and technology solutions that bring transparency to the market.

$400 Billion
Annual transaction notional volume.

450+ Accounting
Clients rely on Chatham for hedge accounting and/or derivative valuations.

Web-based Technology Platform
Proprietary ChathamDirect technology platform serves clients worldwide.

400+ Employees
Employee owned, with offices in 6 countries provides an independent, global perspective into the most effective risk management solutions available.
The Setup
Why this solution exists?

Client/Dealer Facing Applications
- Web Pages
- Public API
- Internal Tools

Financial Models
- IR
- FX/Commodities
- Accounting/Regulation

Math Acceleration Library
Requirements for our solution

Quant Side
• Numbers are 100% reproducible
• Intuitive to use
• Required Math functions:
  • Linear Algebra
  • Linear Regression
  • Random Numbers
  • Summary Stats
  • Factorization
  • Interpolation
  • Non-linear Optimization

Developer Side
• Version control
• MKL integration (for CPU)
• Provide GPU acceleration
• Able to perform in a heterogeneous compute cluster
• Written in C#
Why C#?

C# is a full featured language with a large, active community. It is well supported by Microsoft as well as many other 3rd party offerings. Using this allows our developers to be highly productive.

The Quants development also takes advantage of the internal tools, processes and expertise of our Technology Team.
HOLA
Hybrid Object Linear Algebra (HOLA) Library

• Hybrid Objects can be created on the GPU or the host depending on the context at runtime.

• MKL and CUDA accessed via DLLImport

• Delegate Pattern heavily relied on

• C# Using statements relied on to manage resources and set context
Delegate Pattern

Delegator
<<Interface>> Delegation Interface
Concreate Delegate A
Concreate Delegate B

Matrix
<<Interface>> IMatrix
HostMatrix
DeviceMatrix
The C# using Statement

**MSDN** - Provides a convenient syntax that ensures the correct use of IDisposable objects.

**DotNetPerls** - The using block helps manage resources. Conceptually it protects the whole system's resources by specifying the scope of the usage of the resource. The using statement is combined with a type that implements IDisposable.
Computational Scope

```csharp
using (new ComputationScope())
{
    //Do Something Here
}
using (new ComputationScope(TypeCoersionMode.ForceHost)) {}
using (new ComputationScope(TypeCoersionMode.ForceDevice)) {}
using (new ComputationScope(TypeCoersionMode.FavorLargerObject)) {}
using (new ComputationScope(TypeCoersionMode.FavorDevice)) {}
using (var scope = new ComputationScope())
{
    var X = Matrix.Uninitialized(rows, cols);

    X.AsDurable();
    X.AsPinned();
    X.AsUntracked();

    scope.RegisterDisposable(X);
}
```
Example of object creation

Use Case

// Use Host
var X = Matrix.Uninitialized(rows, cols);

// Use Device
var X = Matrix.Uninitialized(rows, cols);
Example of object operation

Use Case

// Use Host
var X = Matrix.Uninitialized(rows, cols);
var Y = Matrix.Uninitialized(cols, rows);

var Z = X*Y;

// Use Device
var X = Matrix.Uninitialized(rows, cols);
var Y = Matrix.Uninitialized(cols, rows);

var Z = X*Y;

Under the Hood

Matrix
Multiply

CBlasWrapper
dgemm

MKLBlas
dgemm

CuBlas
dgemm
Custom Kernels are Possible too.

Use Case

```csharp
// Use Device
var Y = Matrix.Uninitialized(cols, rows);
var Z = Y.Inv();
```

Under the Hood

```csharp
__global__ void invKernel(double *sourcePtr, double *targetPtr, int size)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < size)
    {
        targetPtr[i] = 1.0 / sourcePtr[i];
    }
}
```
Pitfalls/Solution
Challenge 1: Ephemeral Objects

Use Case

```javascript
var C = X*Y*Z;
for (int i = 0; i < 500; i++)
{
    var C = X*Y*Z;
}
```

Q: What happens to the temp Matrix?
A: It stays in memory.

Under the Hood

```javascript
var temp = X*Y
var C = temp*Z
```

![Memory Allocation Graph](graph.png)
Ephemeral Objects revisited

Use Case

```csharp
using (new ComputationScope(
    TypeCoersionMode.ForceDevice))
{
    var C = Matrix.Uninitialized(rows, cols);
    var X = Matrix.Uninitialized(rows, cols);
    var Y = Matrix.Uninitialized(rows, cols);
    var Z = Matrix.Uninitialized(rows, cols);

    for (int i = 0; i < 500; i++)
    {
        using (new ComputationScope())
        {
            C = X*Y*Z;
        }
    }
}
```

Q: What happens to the temp Matrix?

A: It is disposed as the inner Scope closes.
Ephemeral Objects (cont.)

![Graph showing memory allocation for different matrix sizes and iterations with and without scopes.](image-url)
Challenge 2: Over aggressive compiler optimization

Use Case

(constructed example)

```csharp
var M = Matrix.Zero(rows, cols);
var stats = new SummaryStats(M);
stats.AddMean().AddCorrelation().Compute();
var result = stats.Correlation;
```

Under the Hood

.NET will ‘finalize’ M here *(sometimes)* because it is no longer used by the managed code.
Over aggressive compiler optimization

Use Case

```csharp
using (new ComputationScope())
{
    var M = Matrix.Zero(rows, cols);
    var stats = new SummaryStats(M);
    stats.AddMean().AddCorrelation().Compute();
    var result = stats.Correlation;
}
```

Under the Hood

The Computation Scope is now responsible for letting the compiler know when a Matrix can be released. M will no longer be finalized unexpectedly.

`SummaryStats` is also managed by the computation scope.
Performance
Performance

![Graph showing performance of Exp against matrix size (n x n) with data points for 2x E5620 and Tesla C2050.]
Performance (cont.)

MRG32K3A

Number Generated per Second

Matrix Size (n x n)

- 2x E5620
- Tesla C2050
Performance (cont.)

![Graph showing the performance of matrix multiplication (GEMM) for different matrix sizes (n x n) comparing a 2x E5620 and Tesla C2050, with GFLOPS on the y-axis and Matrix Size (n x n) on the x-axis. The graph shows an increase in GFLOPS as matrix size increases, with a peak and then leveling off for both processors.]

- **GFLOPS**
- **Matrix Size (n x n)**
- **2x E5620**
- **Tesla C2050**
Performance (cont.)

3 Currency FX Simulation

- Millions of Simulation Steps per Second
- Simulation Paths

- 2x E5620
- Tesla C2050
Take-Aways

• HOLA is a complete math Library that is ready to be used in a variety of financial models under many different load and scaling profiles.

• HOLA can be used to easily accelerate math intensive code using GPUs while minimizing the strain/training placed on Quants and Developers.

• HOLA provides consistent numeric stability needed in many different disciplines including Finance.
Interested in HOLA?

HOLA@chathamfinancial.com
Thank You

Ben Eimer
Quantitative Developer
beimer@chathamfinancial.com

Other people on the project:
• Ryan Deering
• John Bagatta
• Liang Cao
• Ethan Giordano
• Maciej Rys
• Jaroslaw Stachowiak
• Piotr Stecko
• Sam Campbell