The Waverun Programming Model for FPGA-Based Heterogeneous Architectures

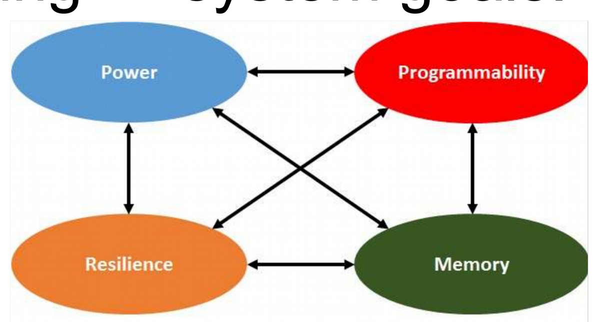
Abhi D.R. and Ron Sass

{adevalap,rsass}@uncc.edu

Reconfigurable Computing Systems Lab: http://www.rcs.uncc.edu

Motivation

The PyDAC runtime for a FPGA-based heterogeneous architecture demonstrated that algorithms using the Divide-and-Conquer design pattern can balance multiple --- sometimes conflicting --- system goals.



Tension among multiple goals

Key Question:

Can the same success be extended to another class of applications that use, for example, the Wavefront design pattern?

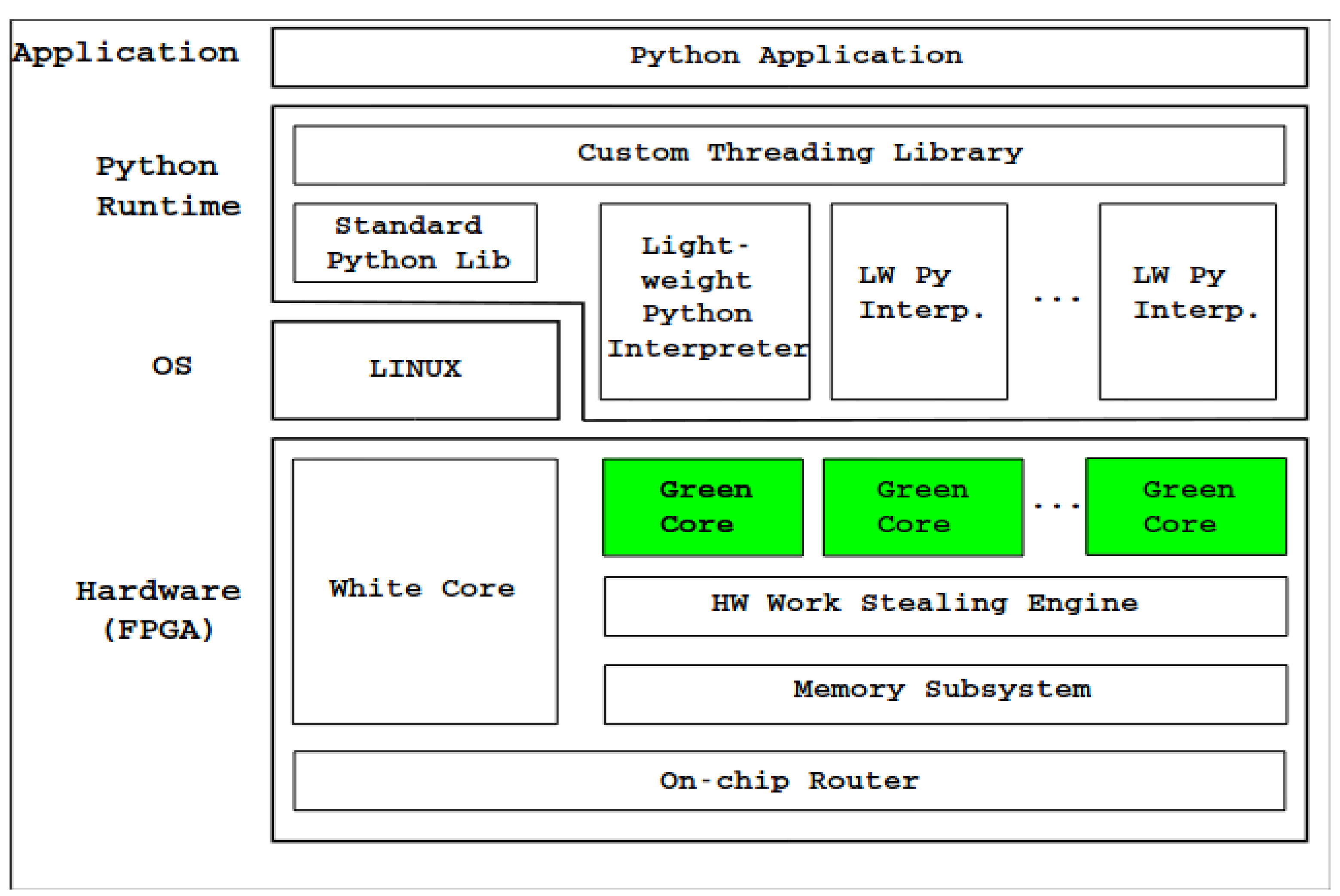
To test this, we left the hardware unchanged and investigated four well-known wavefront algorithms: LU decomposition, QR decomposition, Cholesky decomposition and matrix-matrix multiplication.

Success is defined as:

- ♦ A simple application/runtime interface
- ♦ Efficient use of the memory subsystem

(The hardware does not change so power and resilience are not impacted by these tests.)

FPGA Heterogeneous Architecture



- ♦ **`Match`** and **`Post`** method for memory access
- ♦ Object store and no linear array of memory

Currently implemented in software but part of heterogeneous hardware

Examples

```python
class dpbtrf(waverun):
        def com_corner(self,Ainput,Sdatajk,Sdatakk):
                return Ainput-sum([(p**2*q) for p,q in zip(Sdatajk,Sdatakk)])
        def com_edge(self,Ainput,Sdatajk,Sdatakk,Sdataik,chofact):
                return (Ainput-sum([(p**2*q*r) for p,q,r in zip(Sdatajk,Sdatakk,\
                Sdataik)]))/chofact

        def Corner(self,index,step):
                Res=self.com_corner(Ainput,jkdata,kkdata)#Compute
                self.dptrb_post(index,step,Res)#Post data
                return
        def Edge(self,index,step):
                Res=self.com_edge(Ainput,jkdata,kkdata,ikdata,chofact)#Compute
                self.dptrb_post(index,step,Res)#Post data
                return

        def getFunc(self):
                Func={'00':self.Corner,'0*':self.Edge}
                return Func
        def primer(self):
                for i in range (self.data.shape[0]/block):
                        for j in range (self.data.shape[1]/block):
                                if(i<=j):
                                        self.post(self.data[i][j],'Ainput',[i+1,j+1],0)
                                if(i==0):
                                        self.post(self.data.shape[0]-j-1,'maxi',[i+1,j+1],0)
                                else:
                                        self.post(-1,'maxi',[i+1,j+1],0)
                                        self.post(0,'Result',[i+1,j+1],0)
```

Figure 2: Cholesky Decomposition

$$\begin{bmatrix} C & E & E & E & E & \dots & E & E & E & E \\ & C & E & E & E & \dots & E & E & E & E \\ & & C & E & E & \dots & E & E & E & E \\ & & & & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & & C & E & E \\ & & & & & & & & C & E \\ & & & & & & & & & C \end{bmatrix}$$

C - Corner
E - Edge

Computation:

- Corner

$$D_{jj} = A_{jj} - \sum_{k=1}^{j-1} U_{jk}^2 D_{kk}$$

- Edge

$$U_{ij} = A_{jj} - \sum_{k=1}^{j-1} U_{ik} U_{jk} D_{kk}$$

```python
class dgemm(waverun):
        def Mul(self,Ainput,Binput,Cold):
                return np.add(np.dot(Ainput,Binput),Cold)

        def Corner(self,index,step):
                #Match data
                Cnew=self.Mul(Ainput,Binput,Cold)
                #Post results
                return
        def Xedge(self,index,step):
                #Match data
                Cnew=self.Mul(Ainput,Binput,Cold)
                #Post results
                return
        def Yedge(self,index,step):
                #Match data
                Cnew=self.Mul(Ainput,Binput,Cold)
                #Post results
                return
        def Interior(self,index,step):
                #Match data
                Cnew=self.Mul(Ainput,Binput,Cold)
                #Post results
                return

        def getFunc(self):
                Func={'00':self.Corner,'*0':self.Xedge,'0*':self.Yedge,'**':self.Interior}
                return Func

        def primer(self):
                for i in range (self.data[0].shape[0]/block):
                        for j in range (self.data[1].shape[1]/block):
                                #Post input data
```

$$\begin{bmatrix} C & X & X & X & X & \dots & X & X & X & X \\ Y & I & I & I & I & \dots & I & I & I & I \\ Y & I & I & I & I & \dots & I & I & I & I \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ Y & I & I & I & I & \dots & I & I & I & I \\ Y & I & I & I & I & \dots & I & I & I & I \\ Y & I & I & I & I & \dots & I & I & I & I \end{bmatrix}$$

C - Corner
X - Xedge
Y - Yedge
I - Interior

Figure 1: Matrix Multiplication

Runtime

- ♦ Provides simple user interface
  `Schedule` and **`Gather`** methods to schedule the task and gather the result
- ♦ One program for different types of cores
- ♦ Manages memory keys

Programmer responsibility:

- ♦ Simple object-oriented interface to runtime
- ♦ Provide function based on the index
- ♦ Block of input data via Python method

Results

A simple application/runtime interface with two user methods is implemented

Cholesky Decomposition

```python
Amat=np.random.randint(num,size=(row,column))
myapp=dpbtrf(Amat,block)
myapp.schedule(row/block)
myapp.gather('Result')
```

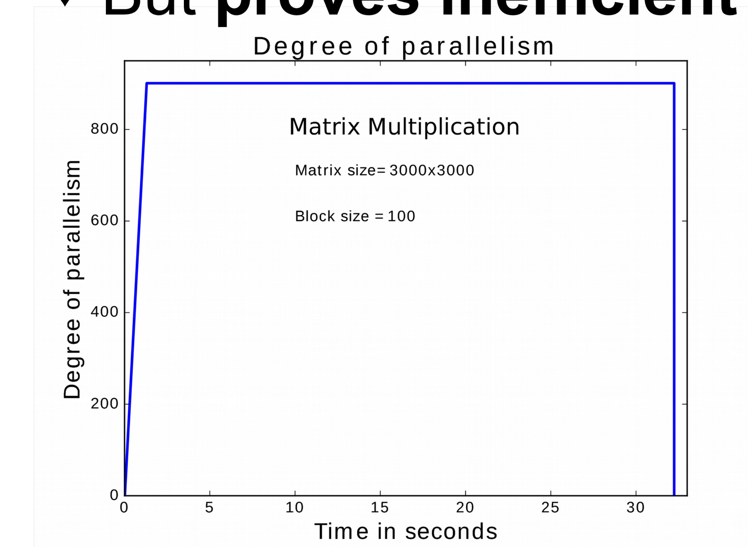
Matrix Multiplication

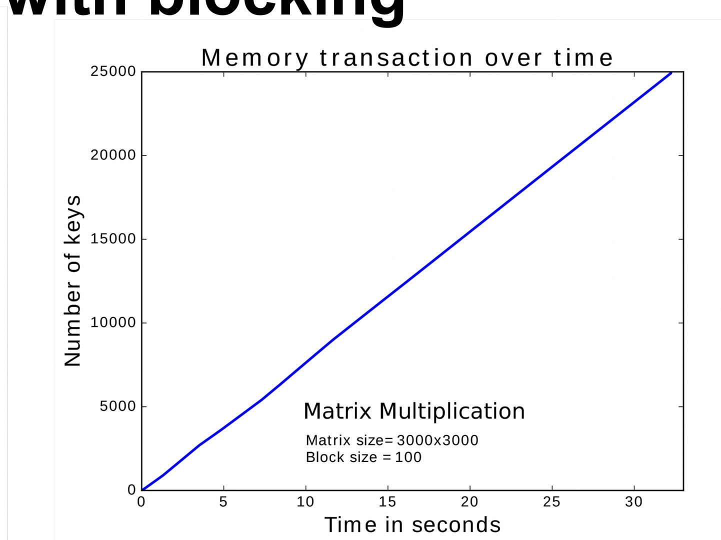
```python

Amat=np.random.randint(num,size=(row,column))
Bmat=np.random.randint(num,size=(row,column))
myapp=dgemm((Amat,Bmat),block)
myapp.schedule(row/block)
myapp.gather('Result')
```

User application using the runtime interface

- ♦ Cholesky, LU, QR, and Matrix multiplication algorithms were implemented using "wavefront" design pattern
- ♦ Cholesky, LU, and QR works well with point-point computation
- ♦ But **proves inefficient with blocking**





Results of matrix multiplication with blocking

Hardware memory subsystem is tested for transaction bandwidth of range 20k-40k. The runtime efficiently uses the system by generating keys of similar range.

Background

Pydac previously implemented:

- ♦ Divide and conquer
- ♦ Proven to be efficient for certain algorithms

B. Huang, R. Sass, N. Debardeleben and S. Blanchard, "Harnessing Unreliable Cores in Heterogeneous Architecture: The PyDac Programming Model and Runtime," *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Atlanta, GA, 2014, pp. 744-749.

Design Patterns

Intel TBB (Parallel patterns)

- ♦ Divide and Conquer
- ♦ Wavefront
- ♦ Agglomeration
- ♦ Elementwise
- ♦ Reduction

Future Work

- ♦ Developing graph related algorithms for waverun
- ♦ Testing the runtime with the heterogeneous hardware