**Multiscale dataflow computing**

**Definition: “Multiscale”**
Problems which have important features at multiple scales

<table>
<thead>
<tr>
<th>Multiple scales of computing</th>
<th>Important features for optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete system level</td>
<td>⇒ balance compute, storage and IO</td>
</tr>
<tr>
<td>parallel node level</td>
<td>⇒ maximize utilization of compute and interconnect</td>
</tr>
<tr>
<td>microarchitecture level</td>
<td>⇒ minimize data movement</td>
</tr>
</tbody>
</table>
| arithmetic level             | ⇒ tradeoff range, precision and accuracy  
    = discretize in time, space and value |
| bit level                    | ⇒ encode and add redundancy         |
| transistor level             | ⇒ create the illusion of ‘0’ and ‘1’ |
A heterogeneous system

CPUs

Infiniband

Dataflow Engines

Memory

0110101001010010
The Multiscale Dataflow Computer

Multiscale Dataflow Computing Platforms

- Risk Analytics Software Platform
- Scientific Computing Software Platform
- Trading Transactions Software Platform

Three Software Layers

- MaxIDE MaxCompiler
- MaxSkins MatLab, Python, R, Excel, C/C++, Fortran

Linux based MaxelerOS: Realtime communication management

HW layer

Controlflow Box: conventional CPUs

- Fast Interconnect

Dataflow Box: Custom Intelligent Memory System
Multiscale Dataflow Advantage

Acoustic Modelling 25x  Weather Modeling 60x  Trace Processing 22x

Financial Risk 32x  Fluid Flow 30x  Seismic Imaging 29x
Amdahl’s Laws

- **First law:** Serial processors always win; too much time spent in programming the parallel processor.
- **Second law:** Fraction of serial code, $s$, limits speedup to:
  
  $$ Sp = \frac{T_1}{T_1(s) + T_1(1-s)/p} \text{ or } Sp = \frac{1}{s + (1-s)/p} $$

Gene Amdahl
Slotnick’s law (of effort)

“The parallel approach to computing does require that some original thinking be done about numerical analysis and data management in order to secure efficient use. In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage.”

-Daniel Slotnick
Dataflow Application Areas
Finance, Geophysics, Chemistry
Physics, Genetics, Astronomy

Application Programming Interface
MaxCompiler: Dataflow in Space and Time
Heterogeneous dataflow+controlflow optimization

Transactions Management
MaxelerOS manages dataflow transaction
MaxelerOS keeps Dataflow-Controlflow balance

Architecture: Static Dataflow
Static Dataflow microarchitecture, cards, boxes
Predictable execution time and efficiency
Control flow vs. Dataflow
Static Dataflow

“Systolic Arrays” without nearest neighbour interconnect restrictions

Static ultradeep (>1000 stage) computing pipelines

One result per clock cycle
### The data movement challenge

<table>
<thead>
<tr>
<th></th>
<th>Today</th>
<th>2018-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double precision FLOP</td>
<td>100pj</td>
<td>10pj</td>
</tr>
<tr>
<td>Moving data on-chip: 1mm</td>
<td>6pj</td>
<td></td>
</tr>
<tr>
<td>Moving data on-chip: 20mm</td>
<td></td>
<td>120pj</td>
</tr>
<tr>
<td>Moving data to off-chip memory</td>
<td>5000pj</td>
<td>2000pj</td>
</tr>
</tbody>
</table>

- Moving data on-chip will use as much energy as computing with it
- Moving data off-chip will use **200x more energy**!
  - And is much slower as well
The memory hierarchy (challenge)

John von Neumann, 1946:

“We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding, but which is less quickly accessible.”
Vertical Co-design of applications

- Deploy domain expertise to **co-design** application, algorithm and computer architecture
17 \times 24 = ?
Thinking Fast and Slow

Daniel Kahneman
Nobel Prize in Economics, 2002

back to 17 × 24

Kahneman splits thinking into:
System 1: fast, hard to control ... 400
System 2: slow, easier to control ... 408
Putting it all together on the arithmetic level

Computing $f(x)$ in the range $[a,b]$ with $|E| \leq 2^{-n}$

**Table**  
**Table+Arithmetic**  
**Arithmetic**

and $+,-,\times,\div$

Tradeoff: number of coefficients, number of bits per coefficient, range versus precision of result and maximal versus average error of result

Dong-U Lee, Altaf Abdul Gaffar, Oskar Mencer, Wayne Luk  
Optimizing Hardware Function Evaluation  
Given range and precision for the result, what is the optimal table+arithmetic solution?

<table>
<thead>
<tr>
<th>Range [bits]</th>
<th>Precision [bits]</th>
<th>Minimal Latency (Optimized for Latency)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4</td>
<td>sin: ( tp2, 24, ) 78, log: ( po2, 8, ) 30, sqr: ( tp2, 18, ) 912</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>sin: ( tp2, 28, ) 348, log: ( tp2, 12, ) 78, sqr: ( tp2, 24, ) 2400</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>sin: ( tp2, 32, ) 792, log: ( tp2, 15, ) 384, sqr: ( tp2, 26, ) 10368</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>sin: ( tp2, 32, ) 3168, log: ( tp2, 21, ) 1056, sqr: ( tp2, 30, ) 23808</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>sin: ( tp3, 40, ) 7872, log: ( tp2, 24, ) 4800, sqr: ( tp3, 34, ) 17920</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>sin: ( tp2, 42, ) 5504, log: ( tp2, 28, ) 11136, sqr: ( tp2, 39, ) 12800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4</td>
<td>sin: ( tp2, 24, ) 78, log: ( po2, 8, ) 30, sqr: ( tp2, 18, ) 912</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>sin: ( tp2, 28, ) 348, log: ( tp2, 12, ) 78, sqr: ( tp2, 24, ) 2400</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>sin: ( tp2, 32, ) 792, log: ( tp2, 15, ) 384, sqr: ( tp2, 26, ) 10368</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>sin: ( tp2, 32, ) 3168, log: ( tp2, 21, ) 1056, sqr: ( tp2, 30, ) 23808</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>sin: ( tp3, 40, ) 7872, log: ( tp2, 24, ) 4800, sqr: ( tp3, 34, ) 17920</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>sin: ( tp2, 42, ) 5504, log: ( tp2, 28, ) 11136, sqr: ( tp2, 39, ) 12800</td>
</tr>
</tbody>
</table>

---

*MAXELER Technologies*
Architecture Level: Star versus Cube Stencil

More Computation in Less Time?

Local temporal parallelism => Cascading timesteps

Figure 1: 2 Alternative stencil choices.
System level: algorithm vs. resource

Insufficient memory to run recompute scheme on 1 GPU, minimum 2
System level: algorithm vs. resource

- GPUs disk speed limited
- Recompute becomes faster
- Enough GPUs => enough memory to store snapshots => network limited
- CPU + DFE have enough memory to store snapshots, store always faster than recompute
Identify and classify options

Data Access Plans  Code Partitioning  Pareto Optimal Options

Transformations

Try to minimise runtime and development time, while maximising flexibility and precision.
Maxeler Dataflow Computers

**MPC C Series**
- CPUs plus DFEs
- Intel Xeon CPU cores and up to 4 DFEs with 192GB of RAM

**MPC X Series**
- DFEs shared over Infiniband
- Up to 8 DFEs with 384GB of RAM and dynamic allocation of DFEs to CPU servers

**MPC N Series**
- Low latency connectivity
- Intel Xeon CPUs and 1-4 DFEs with up to twelve 40Gbit Ethernet connections

**MaxWorkstation**
- Desktop development system

**MaxCloud**
- On-demand scalable accelerated compute resource, hosted in London
MPC-C500

- 1U Form Factor
- 4x dataflow engines
- 12 Intel Xeon cores
- 96GB DFE RAM
- Up to 192GB CPU RAM
- MaxRing interconnect
- 3x 3.5” hard drives
- Infiniband/10GigE
MPC-X1000

- 8 dataflow engines (384GB RAM)
- High-speed MaxRing
- Zero-copy RDMA between CPUs and DFEs over Infiniband
- Dynamic CPU/DFE balancing
Dataflow clusters

- Optimized to balance resources for particular application challenges
- Flexible at design-time and at run-time

48U seismic imaging cluster

42U in-memory analytics cluster
Application Programming Process

1. **Start**
   - Identify code for acceleration and analyze bottlenecks

2. **Original Application**
   - Transform app, architect and model performance
   - Write MaxCompiler code
   - Integrate with CPU code
   - Simulate DFE

3. **Accelerated Application**
   - Meets performance goals?
   - Build full DFE configuration
   - Functions correctly?

4. **Decision Points**
   - Yes: Continue
   - No: Return to previous step and reevaluate
Programming with MaxCompiler

User Input

SLiC

Host Application (.c, .f)

Computationally intensive components

MaxIDE

Application Kernel(s) (.java)

Manager Configuration (.java)

Kernel Compiler

Manager

MaxelerOS SW HW

Compiler, Linker

HW Accelerator (.max)

Output

Accelerated Application (executable)
Programming with MaxCompiler

```java
package convolve;

import com.maxeler.maxcompiler.v2.kernelcompiler.Kernel;

class ConvolveKernel extends Kernel {
    int main(void) {
        const int size = 364;
        int sizeBytes = size * sizeof(int32_t);
        float *x = malloc(sizeBytes);
        float *y = malloc(sizeBytes);
        float *z = malloc(sizeBytes);
        int coeff_a = 3;

        printf("Generating data...
");
        for(int i = 0; i < size; i++) {
            x[i] = i;
            y[i] = random() % 100;
        }

        printf("Writing x data to DFE LMem...
");
        Convolve_writeMem(0, sizeBytes, x);

        printf("Ranining computation on DFE...
");
        Convolve(coeff_a, size, x);
    }
}
```
for (int i = 0; i < DATA_SIZE; i++)
    y[i] = x[i] * x[i] + 30;

\[ y_i = x_i \times x_i + 30 \]
Programming with MaxCompiler

CPU Code (.c)
#include “MaxSLiCInterface.h"
#include “Calc.max"
int *x, *y;
Calc(x, y, DATA_SIZE)

Manager (.java)
Manager m = new Manager(“Calc”);
Kernel k =
  new MyKernel();
m.setKernel(k);
m.setIO(  
  link(“x”, CPU),
  link(“y”, CPU));
m.createSLiCInterface();
m.build();

MyKernel (.java)
DFEVar x = io.input("x", dfeInt(32));
DFEVar result = x * x + 30;
io.output("y", result, dfeInt(32));
Programming with MaxCompiler

CPU
  CPU Code
  SLiC
  MaxelerOS

Main Memory

Manager (.java)
Manager m = new Manager("Calc");
Kernel k =
  new MyKernel();
m.setKernel(k);
m.setIO(
  link("x", CPU),
  link("y", LMEM_LINEAR1D));
m.createSLiCInterface();
m.build();

MyKernel (.java)
DFEVar x = io.input("x", dfeInt(32));
DFEVar result = x * x + 30;
io.output("y", result, dfeInt(32));

CPUCode (.c)
#include "MaxSLiCInterface.h"
#include "Calc.max"
int *x, *y;

Calc(x, DATA_SIZE)
public class MyKernel extends Kernel {

    public MyKernel (KernelParameters parameters) {
        super(parameters);

        HWVar x = io.input("x", hwInt(32));

        HWVar result = x * x + 30;

        io.output("y", result, hwInt(32));
    }
}
Kernel Streaming

![Diagram showing a flow of elements from 5 to 0, with operations at each step including a multiplication by 'x', an addition with '30', and a final output]

- Elements flow from 5 to 0.
- Each step includes an operation: multiplication by 'x', addition of '30', and then an output.
Kernel Streaming

5 4 3 2 1 0

x

0

x

+

30

y

[Blank boxes]
Kernel Streaming

5 4 3 2 1 0

x

1

x

0

+

30

y

Output
Kernel Streaming

```
5 4 3 2 1 0
```

```
x
```

```
2
```

```
x
```

```
1
```

```
+ 30
```

```
30
```

```
y
```

```
Kernel Streaming
Kernel Streaming

5 4 3 2 1 0
x
4
x
9
30
+
34
y
30 31
Kernel Streaming

5 4 3 2 1 0

30 31 34

30

39

16

x

5

x

y

+
Kernel Streaming
Kernel Streaming
Kernel Streaming

5 4 3 2 1 0

30 31 34 39 46 55
A (slightly) more complex kernel

```java
14  class MovingAverageSimpleKernel extends Kernel {
15
16  MovingAverageSimpleKernel(KernelParameters parameters) {
17    super(parameters);
18
19    DFEVar x = io.input("x", dfeFloat(8, 24));
20
21    DFEVar prev = stream.offset(x, -1);
22    DFEVar next = stream.offset(x, 1);
23    DFEVar sum = prev + x + next;
24    DFEVar result = sum / 3;
25
26    io.output("y", result, dfeFloat(8, 24));
27  }
28}
```
Kernel Execution

5 4 3 2 1 0

-1 0 +1

? ? + 1

? ? + 3

? y

?
Kernel Execution
Kernel Execution
Kernel Execution
Kernel Execution
Real data flow graph as generated by MaxCompiler
4866 nodes;
10,000s of stages/cycles
Maxeler University Program
Tackling a vertical problem at multiple scales can allow you to make major jumps in capability.

Dataflow computing achieves high performance through:
- Explicitly putting data movement at the heart of the program
- Employing massive parallelism at low clock frequencies
- Embodying application co-design

Many scientific applications can benefit from this approach.