Acceleration in the Wild, with Data Flow Computing

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

James Spooner, VP of Acceleration

QCon, Finance Track, 08 March 2012

Acceleration in the Wild with Data Flow

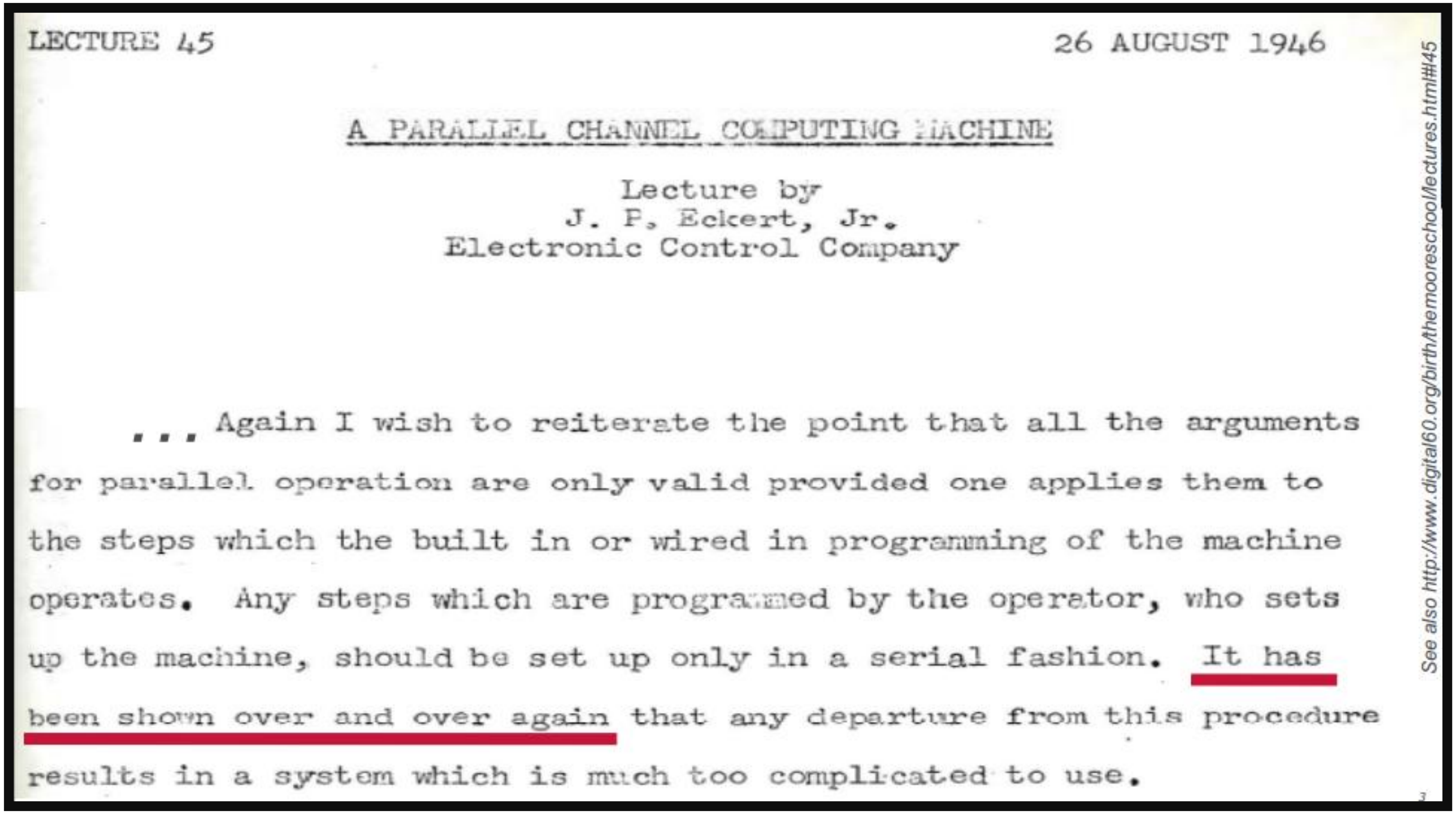
- Deliberate, focused approach to improving application speed
  - Involves adding Data Flow Engines (DFEs)
  - Makes **some** of the program faster
  - Will be programmed *intentionally* and be architecture specific
  - Will exploit as much available **parallelism** as possible
  - May require transformations to **expose** parallelism
  - May have multiple implementations

Maxeler is a acceleration specialist, delivering end-to-end performance for a range of clients in the banking and oil/gas exploration industries.

Making efficient use of Silicon

MAXELER
Technologies
MAXIMUM PERFORMANCE COMPUTING

Computing History…



LECTURE 45 26 AUGUST 1946

A PARALLEL CHANNEL COMPUTING MACHINE

Lecture by
J. P. Eckert, Jr.
Electronic Control Company

. . . Again I wish to reiterate the point that all the arguments for parallel operation are only valid provided one applies them to the steps which the built in or wired in programming of the machine operates. Any steps which are programmed by the operator, who sets up the machine, should be set up only in a serial fashion. It has been shown over and over again that any departure from this procedure results in a system which is much too complicated to use.

See also http://www.digital60.org/birth/themooreschool/lectures.html#45

Credit: Prof. Paul H.J. Kelly

- J. P, Eckert, Jr (Co-Inventor of ENIAC)

MAXELER Technologies

Computing History…

"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 (Chief Architect of ILLIAC IV), 1967

Credit: Prof. Michael J. Flynn

So what happened?

- Eckert (and Amdahl) were right, Slotnik was wrong, until…
- Serial computing hit the wall(s) last decade:
  - The *memory wall*; the increasing gap between processor and memory speeds. This effect pushes cache sizes larger in order to mask the latency of memory. This helps only to the extent that memory bandwidth is not the bottleneck in performance.
  - The *ILP wall*; the increasing difficulty of finding enough parallelism in a single instruction stream to keep a high-performance single-core processor busy.
  - The *power wall*; the trend of consuming exponentially increasing power with each factorial increase of operating frequency. This increase can be mitigated by "shrinking" the processor by using smaller traces for the same logic. The *power wall* poses manufacturing, system design and deployment problems that have not been justified in the face of the diminished gains in performance due to the *memory wall* and *ILP wall*.

$$P_{avg} = C_{load} \cdot {V_{DD}}^2 \cdot f$$

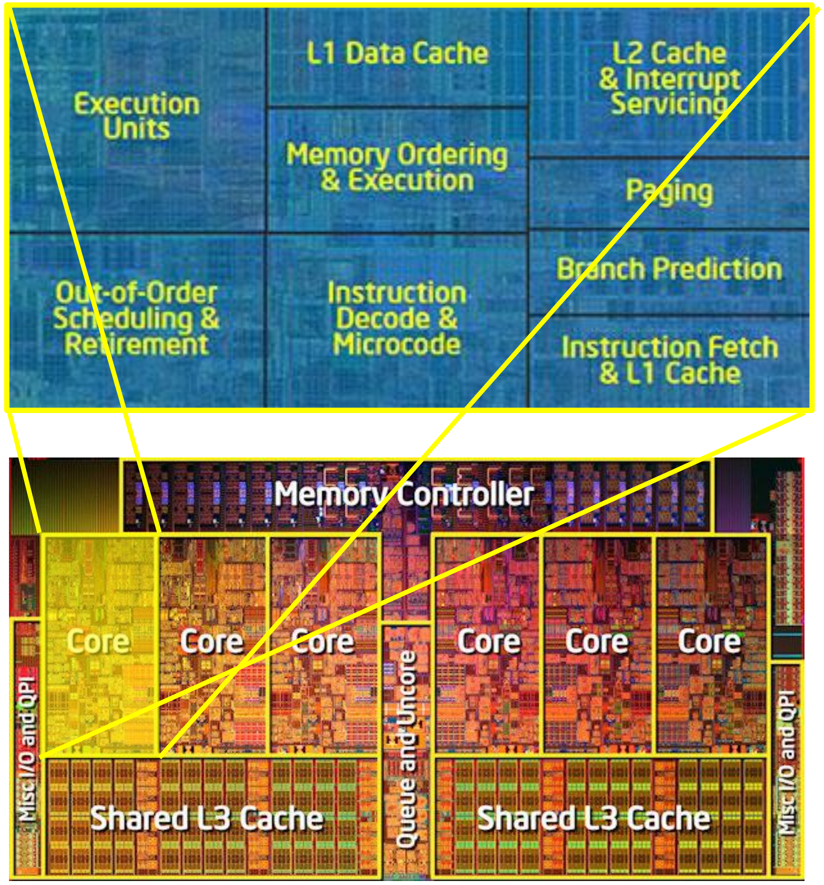
Source: Wikipedia

Using silicon efficiently - parallelism

Level of Parallelism	Examples	Costs
Coarse Grained	- Multi-Node, Multi-chip, multi-core - Process / thread level parallism	-Developing a distributed Distributed system - Locks, mutexes, queues, etc.
Fine Grained	-Instruction level parallelism (ILP) -Out-of-order execution, superscalar, instruction pipelining, speculative execution -Data level parallelism -SIMD / SSE	- Lots of silicon - Compiler can do some work upfront
Ultra Fine Grained	-Data Flow architectures - Massively parallel, lock free, hazard free, streaming datapaths	- Resolve once

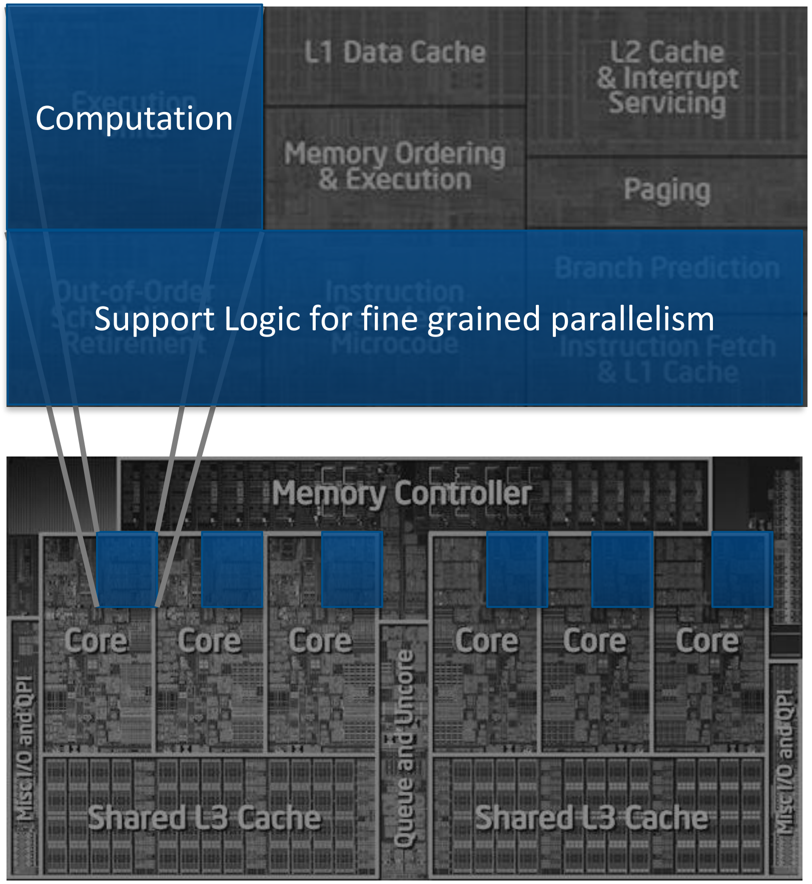
How is modern silicon used?

Intel 6-Core X5680 “Westmere”



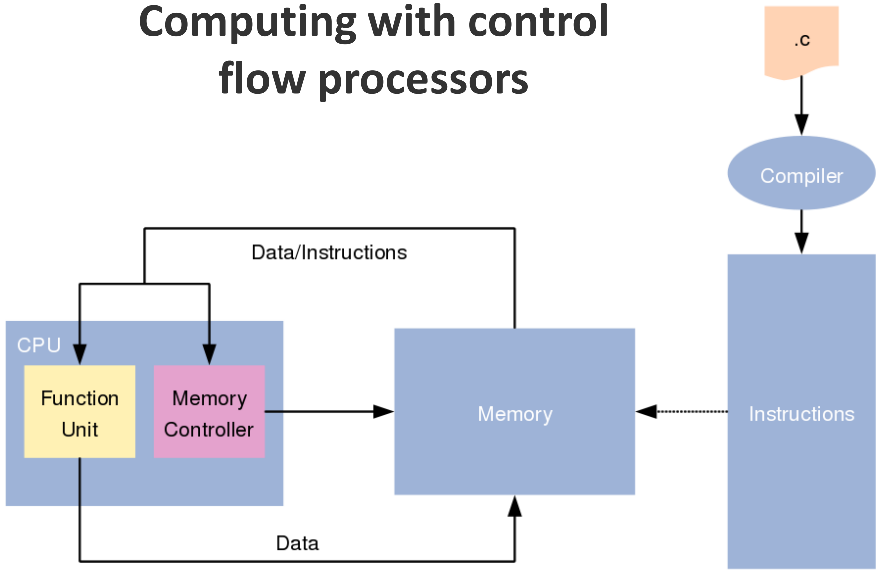
How is modern silicon used?

Intel 6-Core X5680 "Westmere"


Computation
Execution
L1 Data Cache
L2 Cache & Interrupt Servicing
Memory Ordering & Execution
Paging
Out-of-Order Scheduling & Retirement
Instruction Decode & Microcode
Branch Prediction
Instruction Fetch & L1 Cache
Support Logic for fine grained parallelism
Memory Controller
Core
Core
Core
Core
Core
Core
Misc I/O and QPI
Queue and Uncore
Misc I/O and QPI
Shared L3 Cache
Shared L3 Cache

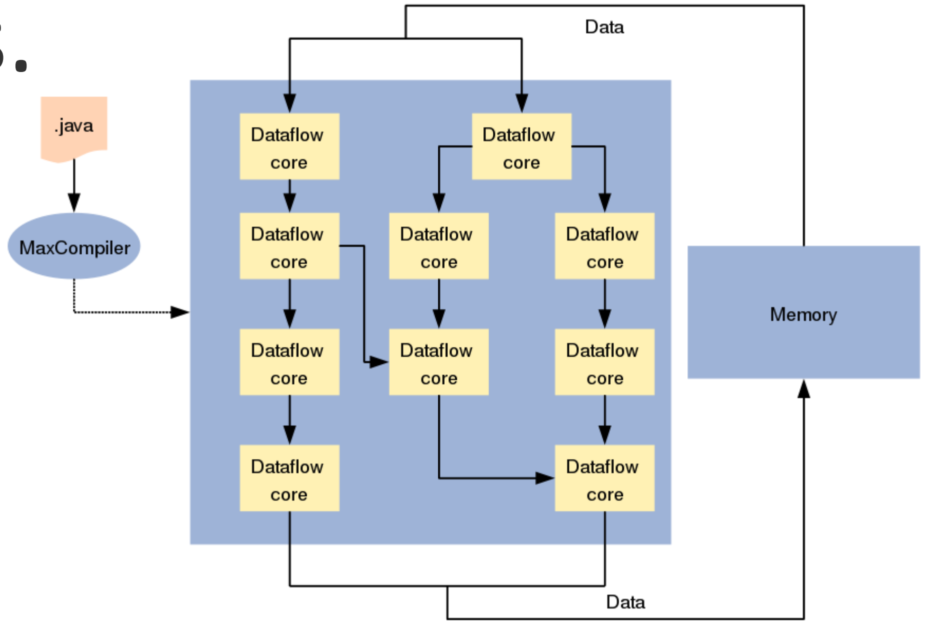
What is Dataflow Computing?

Computing with control flow processors



VS.

Computing with dataflow engines (DFEs)

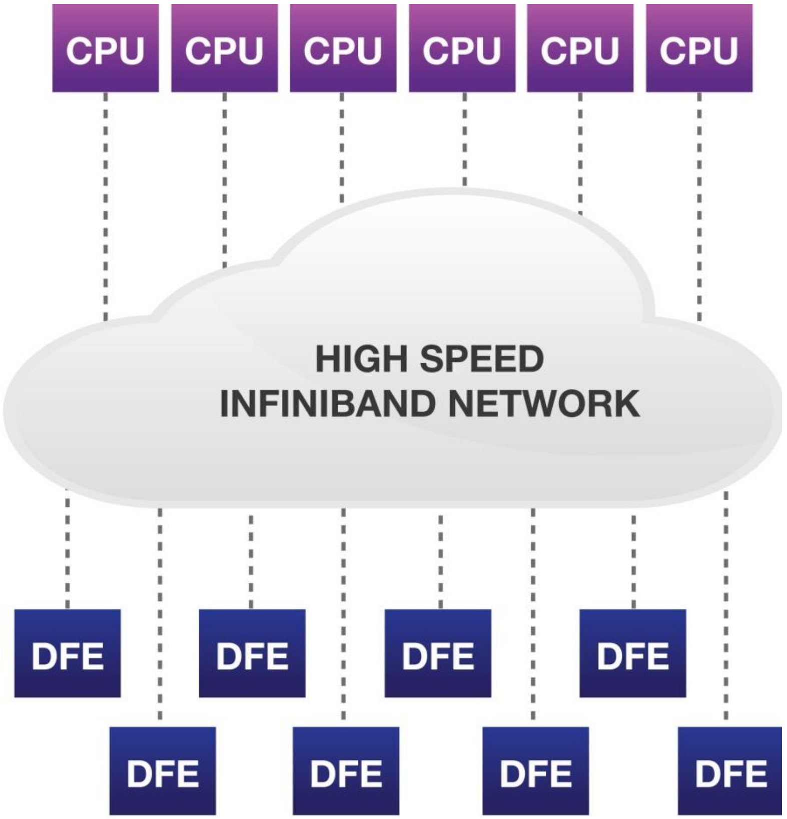


MPC X Series

1U dataflow cloud providing dynamically scalable compute capability over Infiniband

MPC-X1000

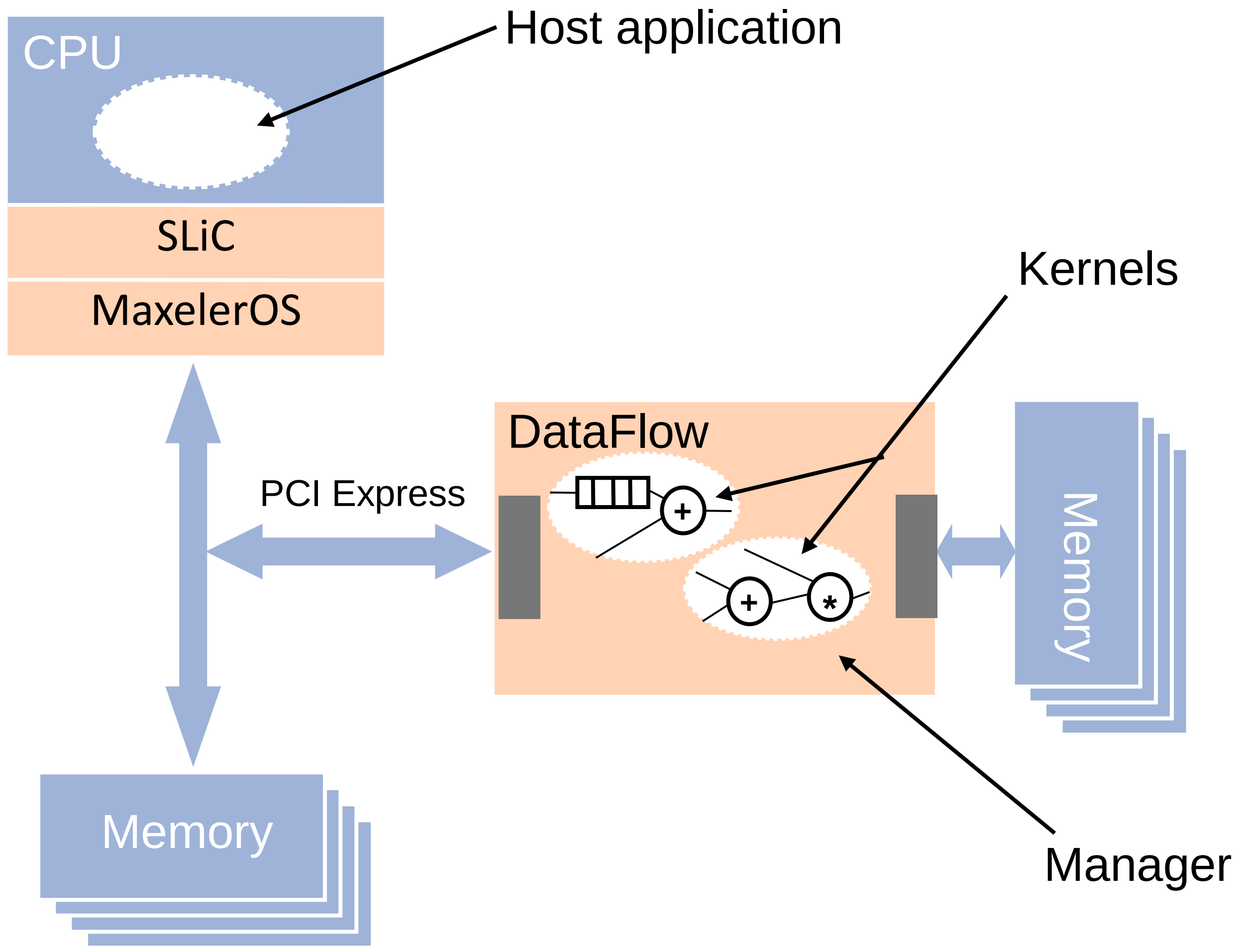
- 8 *vectis* dataflow engines (DFEs)
- 192GB of DFE RAM
- Dynamic allocation of DFEs to conventional CPU servers
  - Zero-copy RDMA between CPUs and DFEs over Infiniband
- Equivalent performance to 40-60 x86 servers



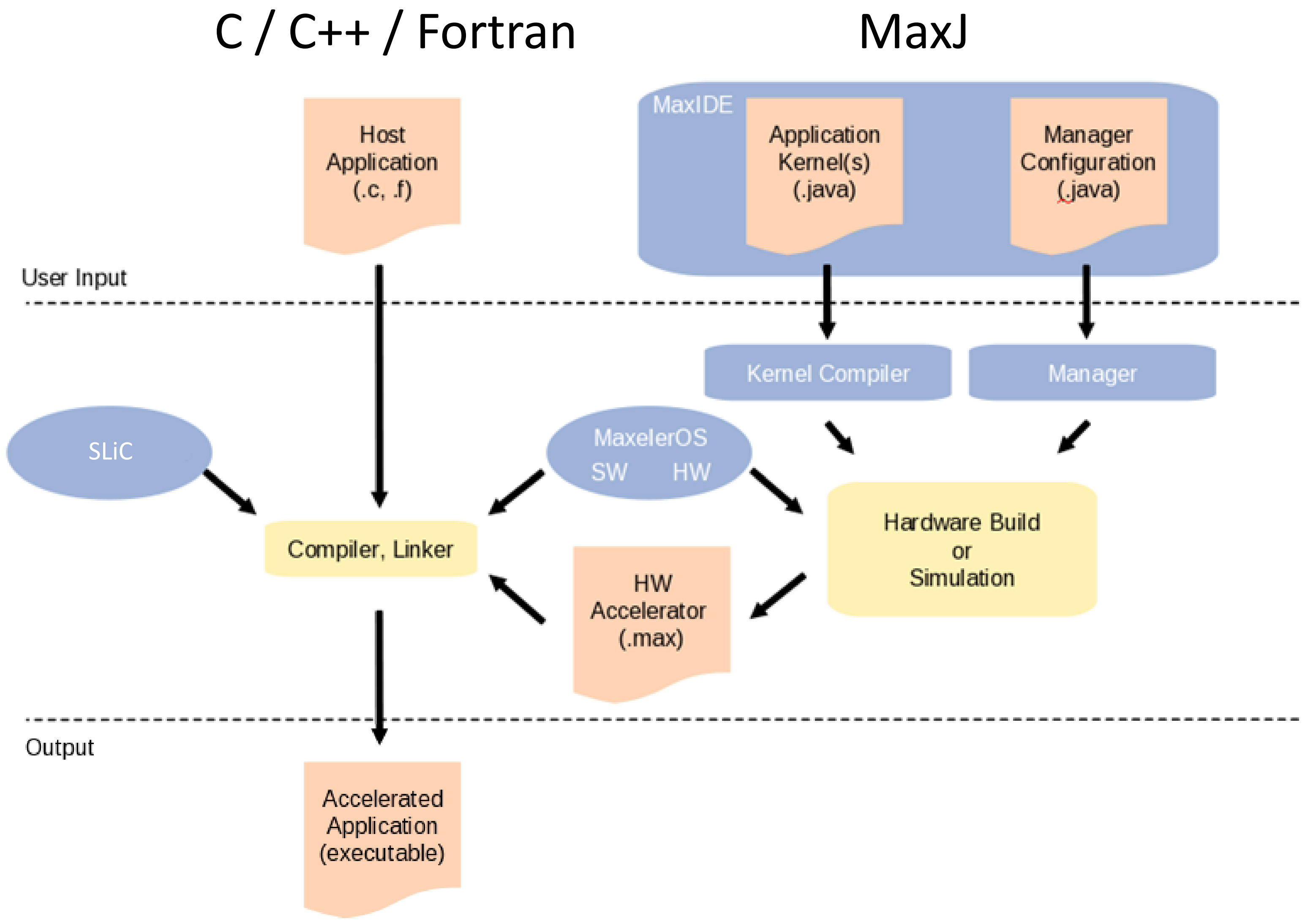
Dataflow Programming

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

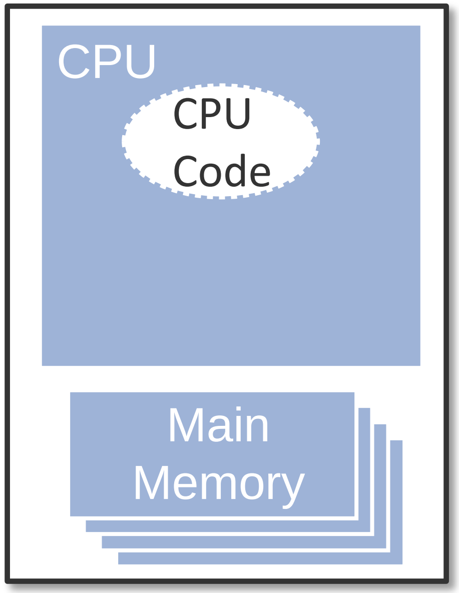
Application Components



Programming with MaxCompiler


C / C++ / Fortran
MaxJ
MaxIDE
Host Application (.c, .f)
Application Kernel(s) (.java)
Manager Configuration (.java)
User Input
Kernel Compiler
Manager
SLiC
MaxelerOS
SW HW
Compiler, Linker
Hardware Build or Simulation
HW Accelerator (.max)
Output
Accelerated Application (executable)

MaxCompiler Development Process

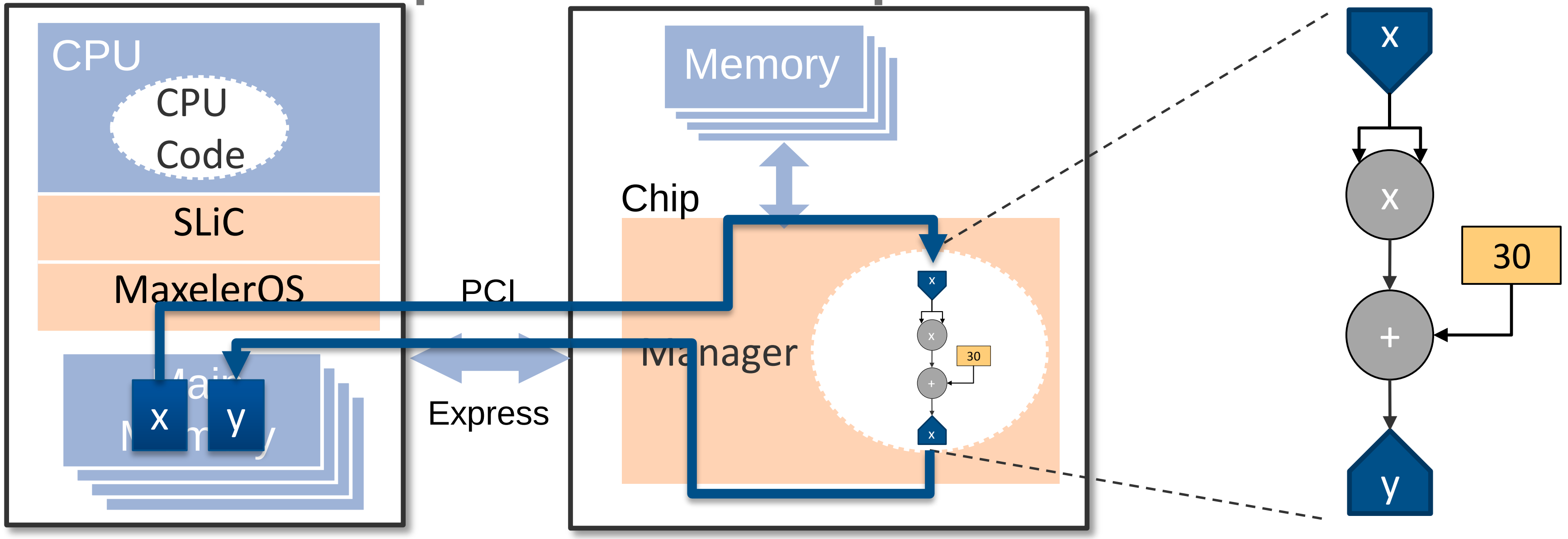


CPU Code (.c)

```
int *x, *y;
for (int i =0; i < DATA_SIZE; i++)
    y[i]= x[i] * x[i] + 30;
```

$$y_i = x_i \times x_i + 30$$

MaxCompiler Development Process



CPU Code (.c)

```
#include "MaxSLiCInterface.h"
#include "Calc.max"
int *x, *y;
for (int i =0; i < DATA_SIZE; i++)
   y[i]= x[i] * x[i] + 30;
Calc(x, y, DATA_SIZE)
```

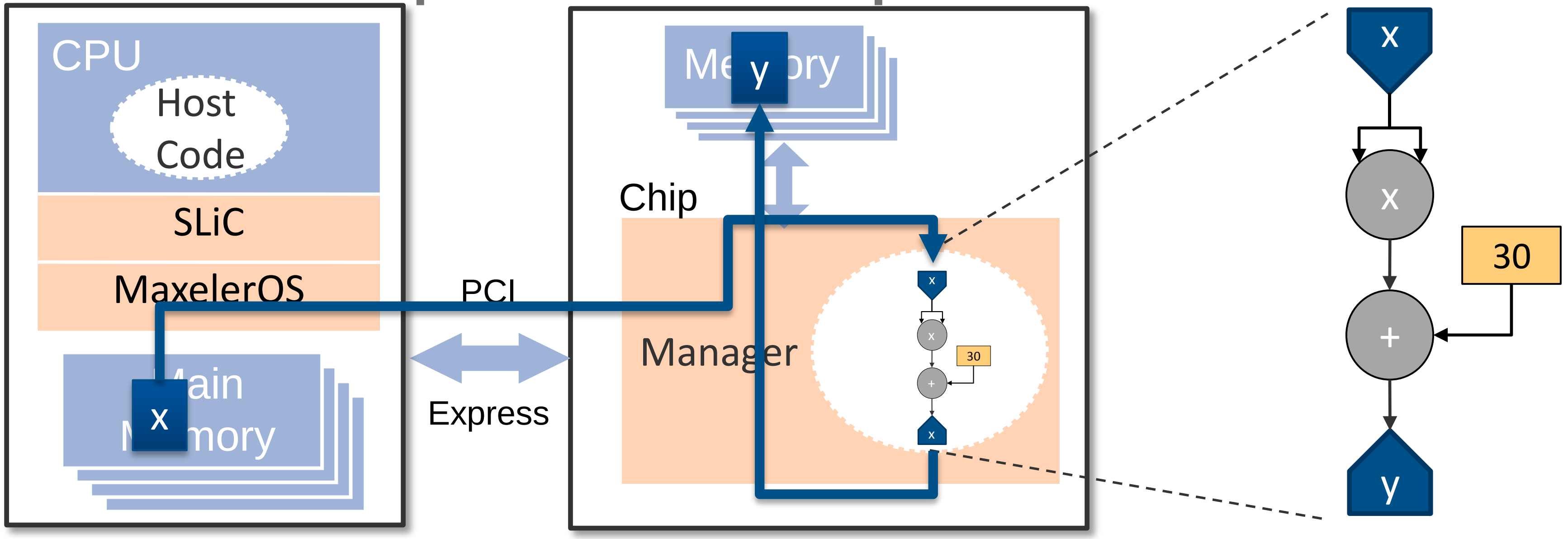
Manager (.java)

```
Manager m = new Manager("Calc");
Kernel k =
    new MyKernel();

m.setKernel(k);
m.setIO(
   link("x", PCIE),
   link("y", PCIE));
m.addMode(modeDefault());
m.build();
```

MyKernel (.java)

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

HWVar result = x * x + 30;

io.output("y", result, hwInt(32));
```

MaxCompiler Development Process



CPUCode (.c)

```
#include “MaxSLiCInterface.h”
#include “Calc.max”
int *x, *y;


Calc(x, DATA_SIZE)
```

Manager (.java)

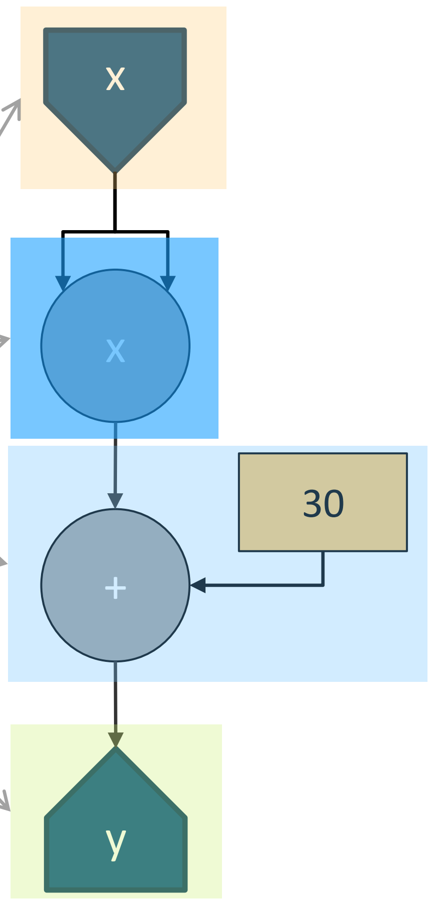
```
Manager m = new Manager();
Kernel k =
    new MyKernel();

m.setKernel(k);
m.setIO(
   link(“x", PCIE),
   link(“y", DRAM_LINEAR1D));
m.addMode(modeDefault());
m.build();
```

MyKernel (.java)

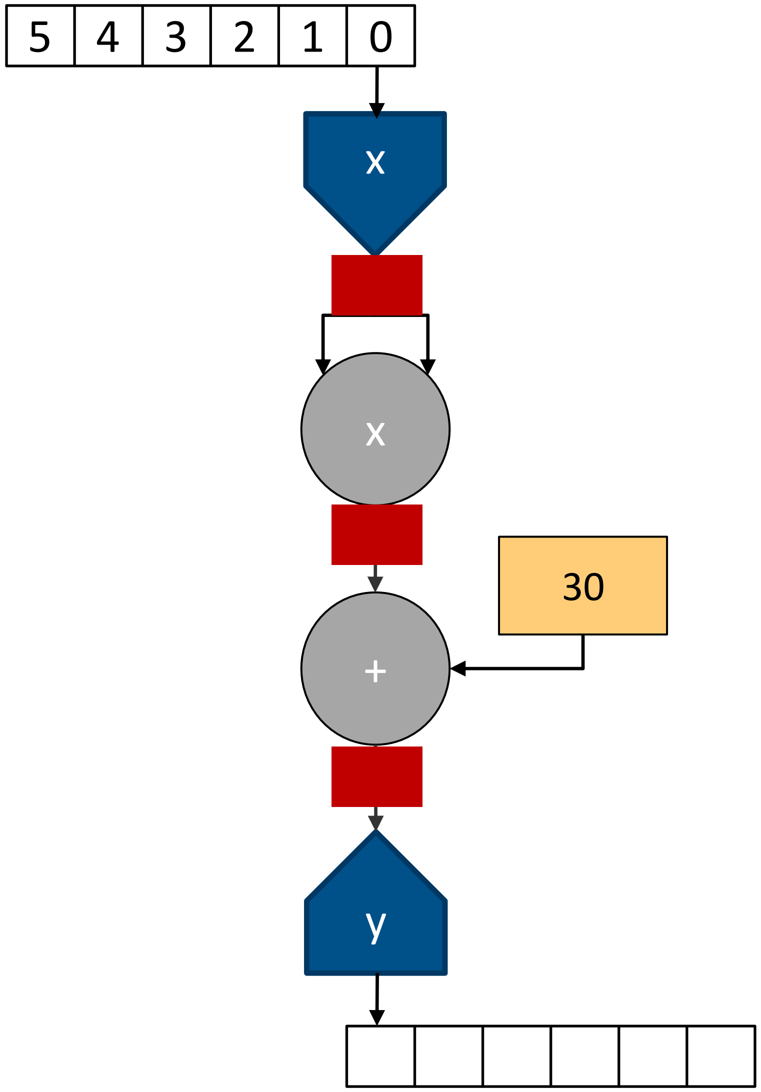
```
HWVar x = io.input("x", hwInt(32));

HWVar result = x * x + 30;

io.output("y", result, hwInt(32));
```

The Full Kernel

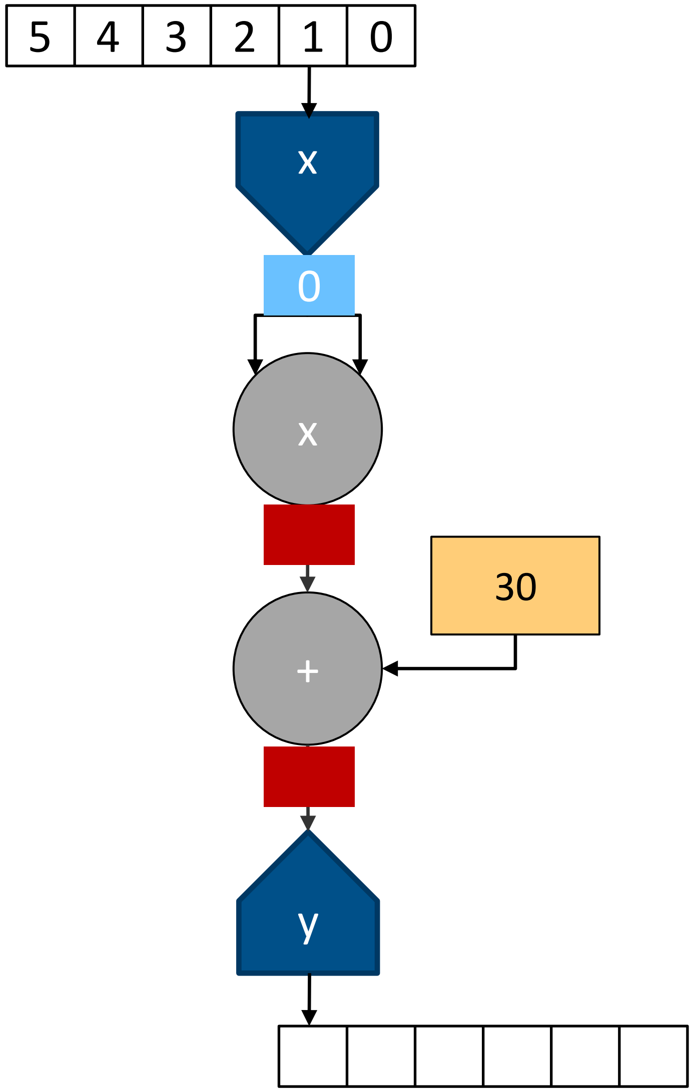
```
public class MyKernel extends Kernel {

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

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

          HWVar result = x * x + 30;

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



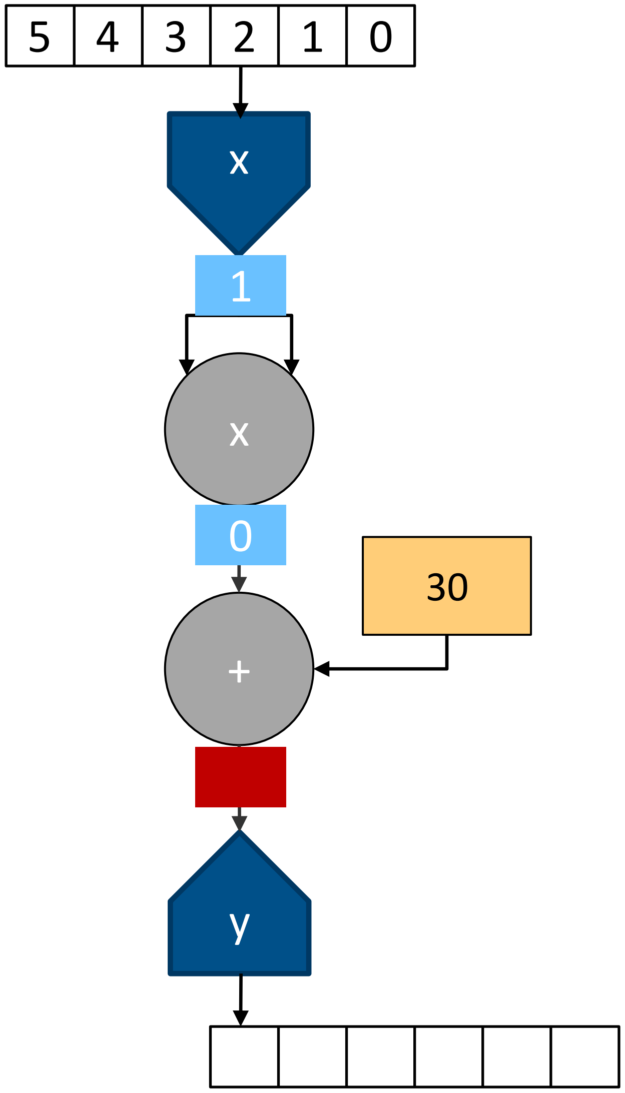
Kernel Streaming: In Hardware



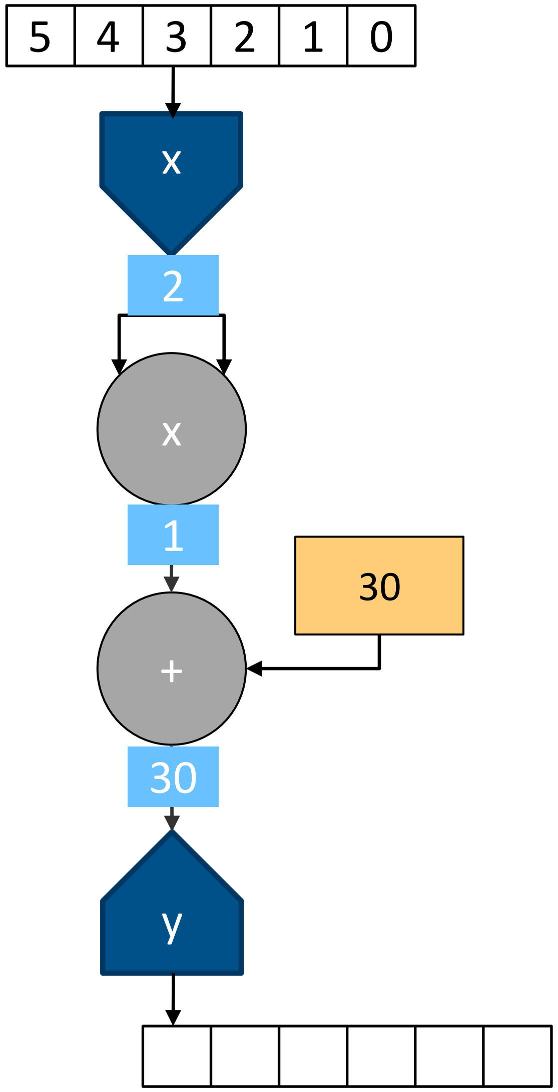
Kernel Streaming: In Hardware



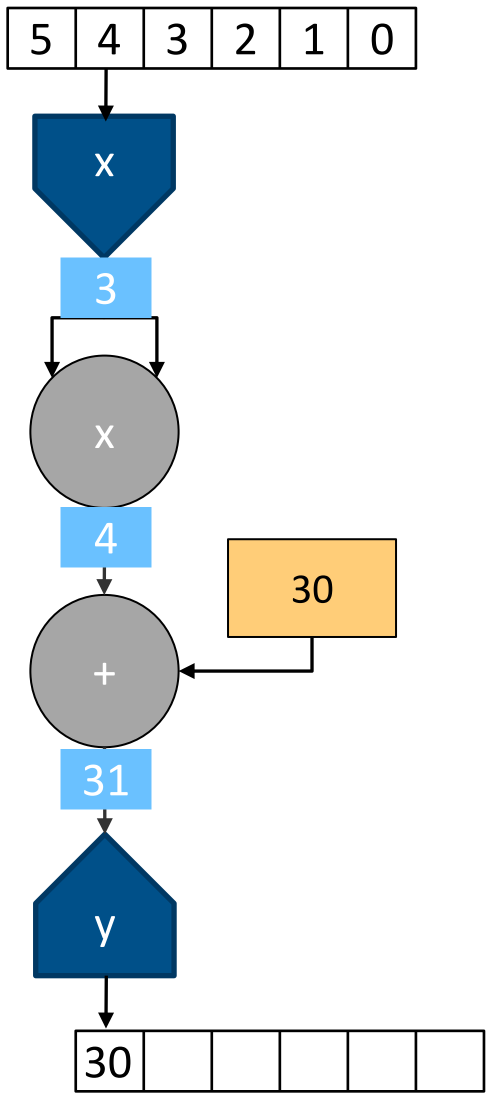
Kernel Streaming: In Hardware



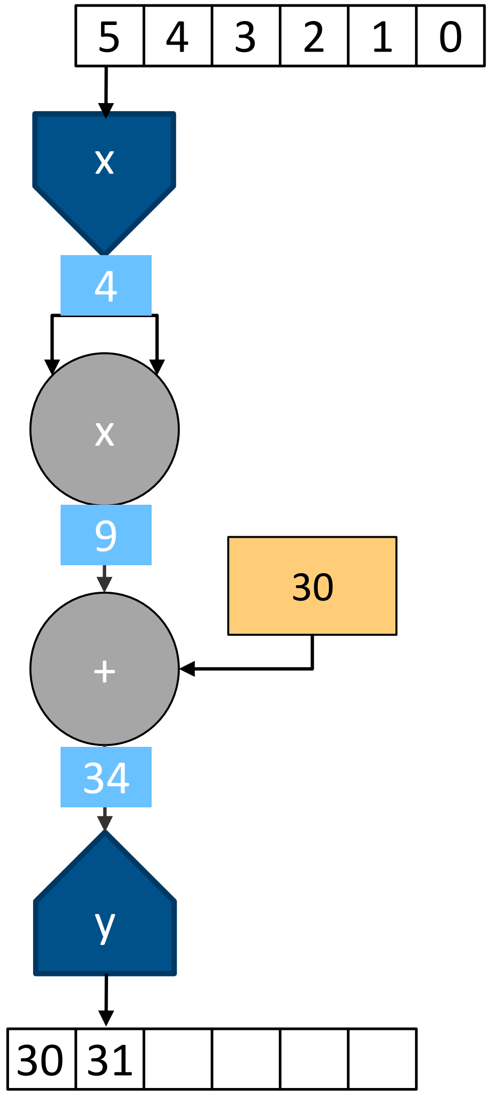
Kernel Streaming: In Hardware



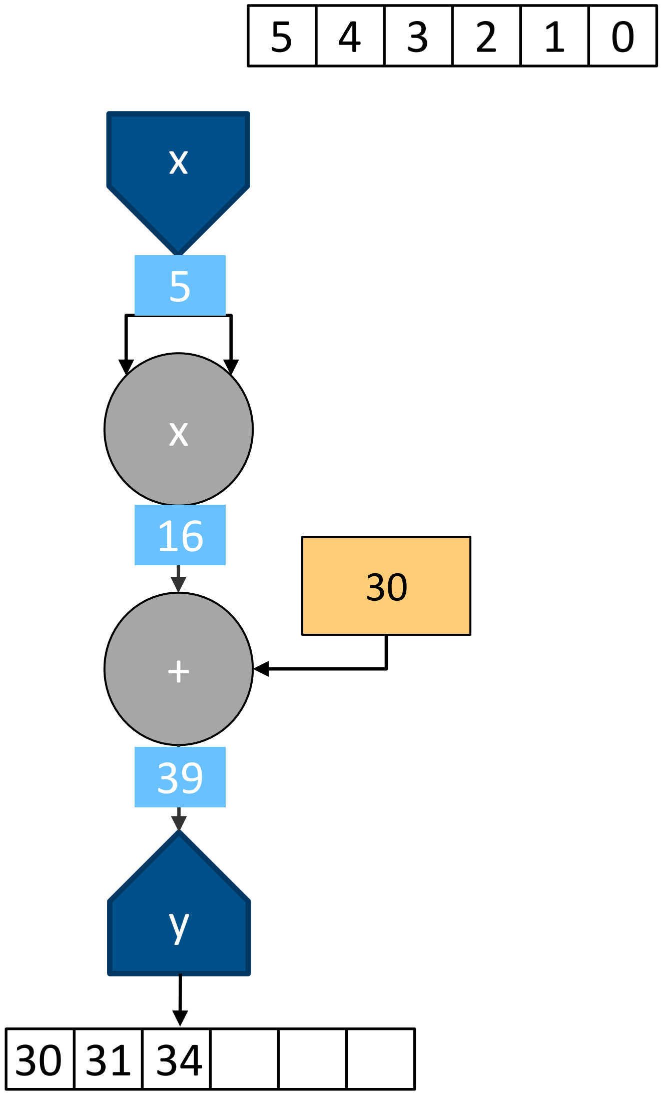
Kernel Streaming: In Hardware



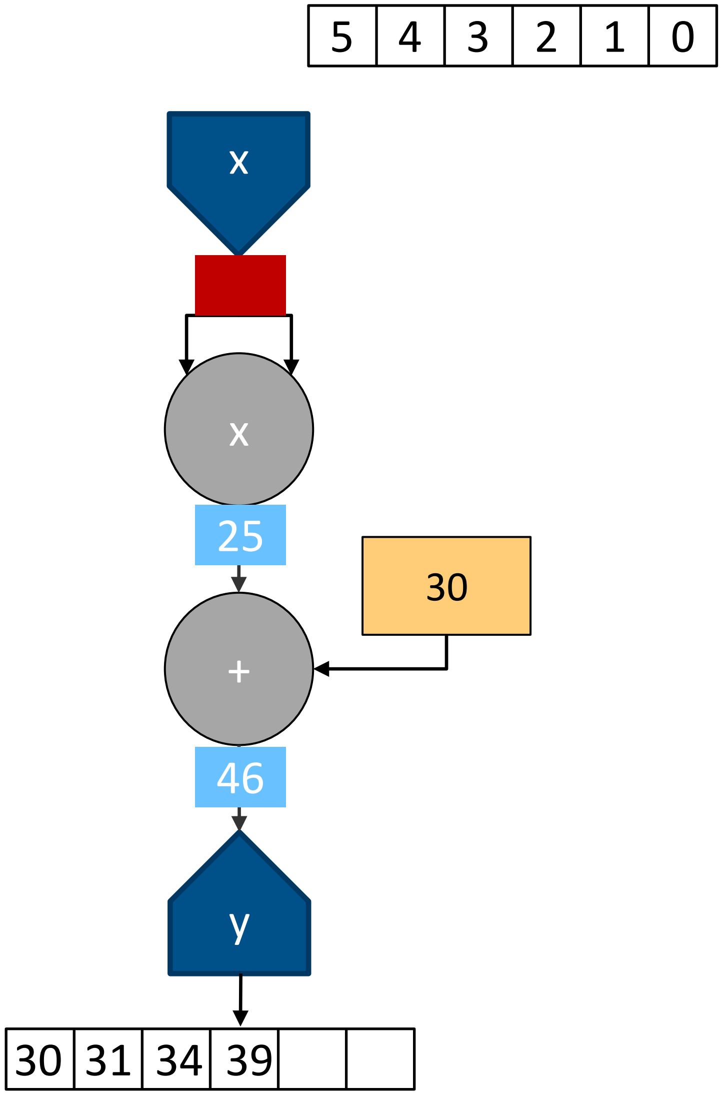
Kernel Streaming: In Hardware



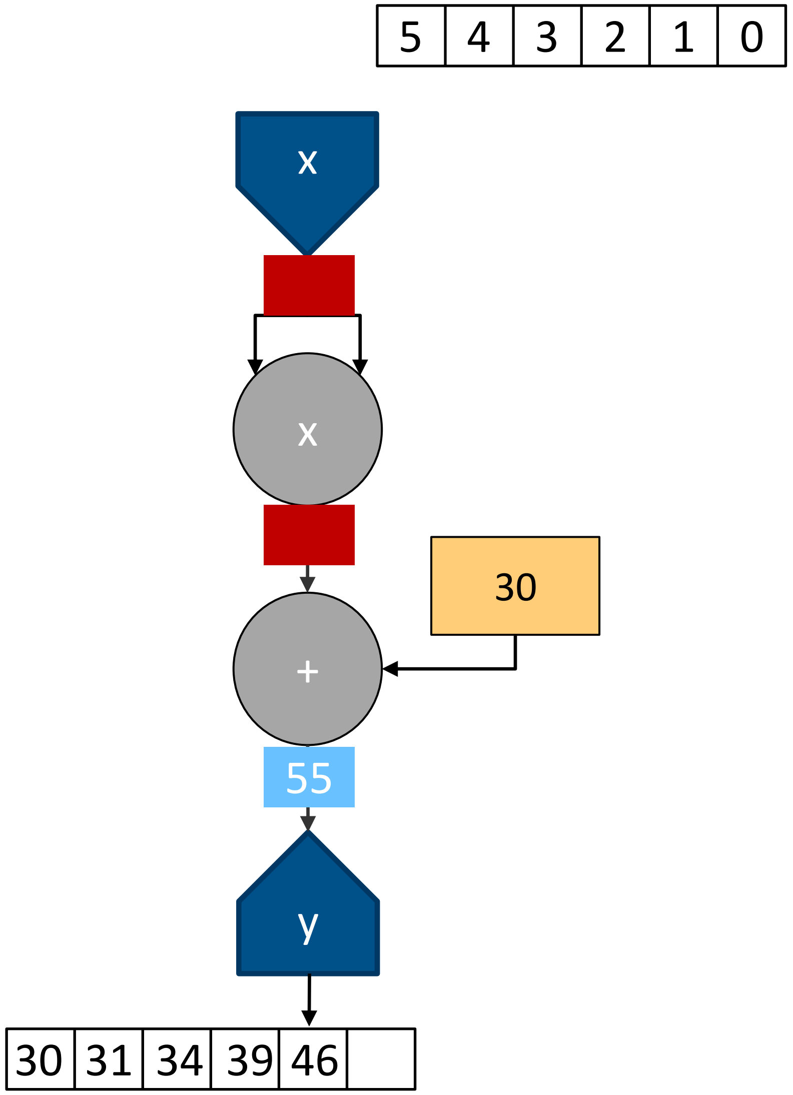
Kernel Streaming: In Hardware



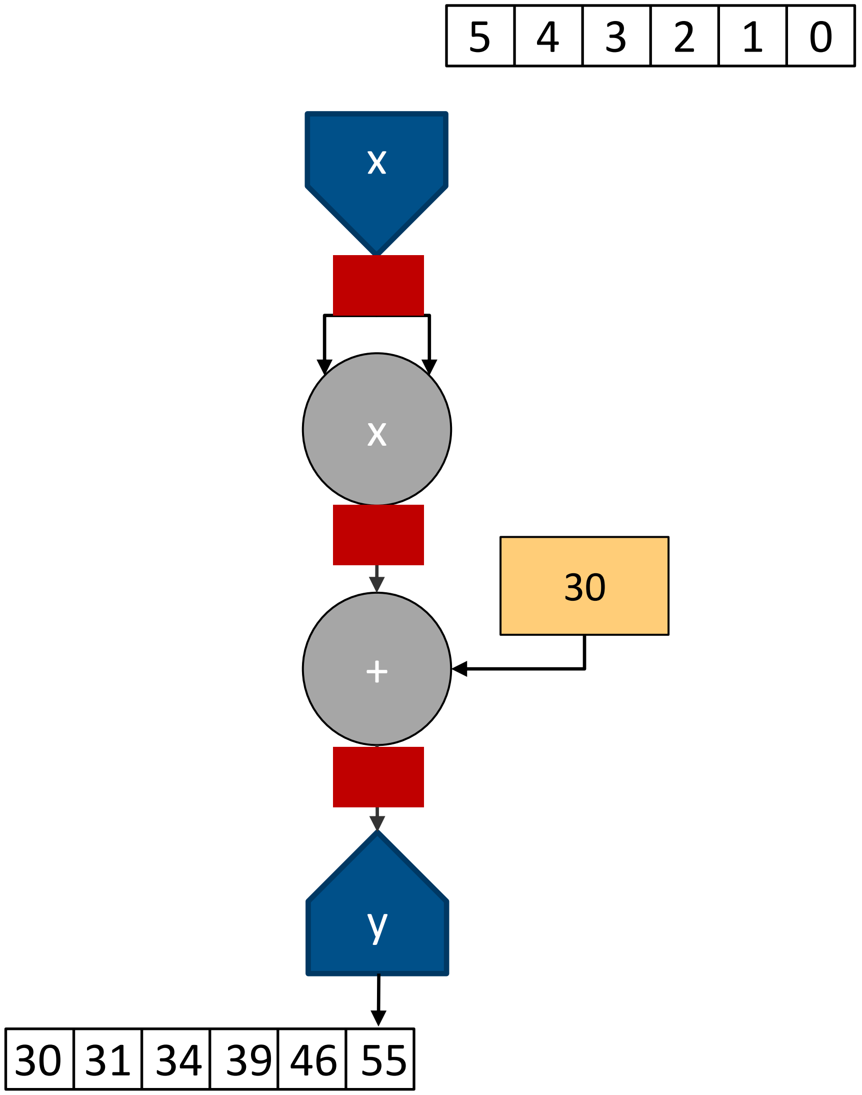
Kernel Streaming: In Hardware



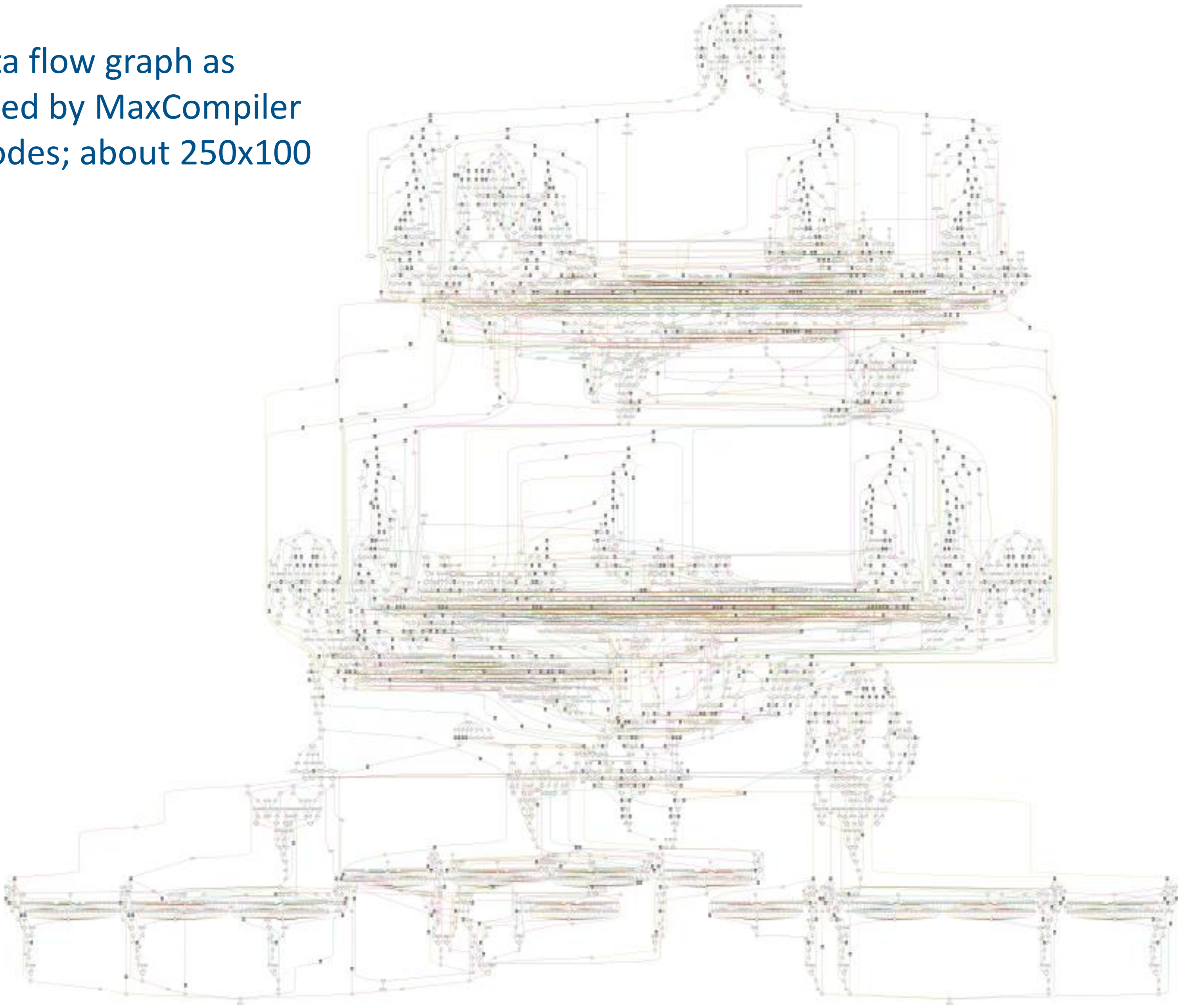
Kernel Streaming: In Hardware



Kernel Streaming: In Hardware



Data flow graph as generated by MaxCompiler 4866 nodes; about 250x100



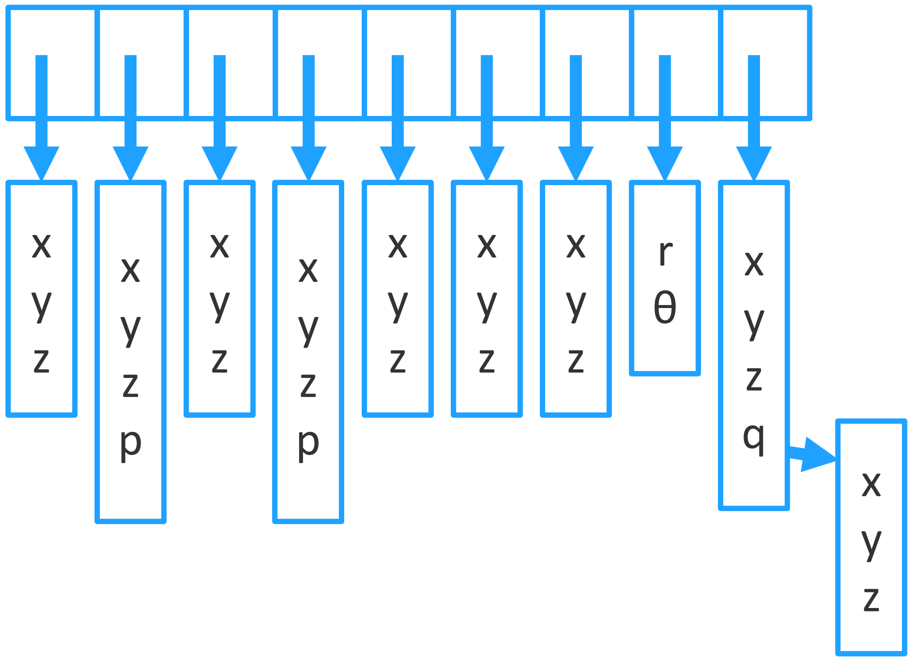
How we approach Acceleration

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

What always makes Acceleration hard?

- Messy code
- Complicated build dependences
- Confused control-flow
- Impenetrable data access
- Pointer-intensive data structures
- Premature optimization

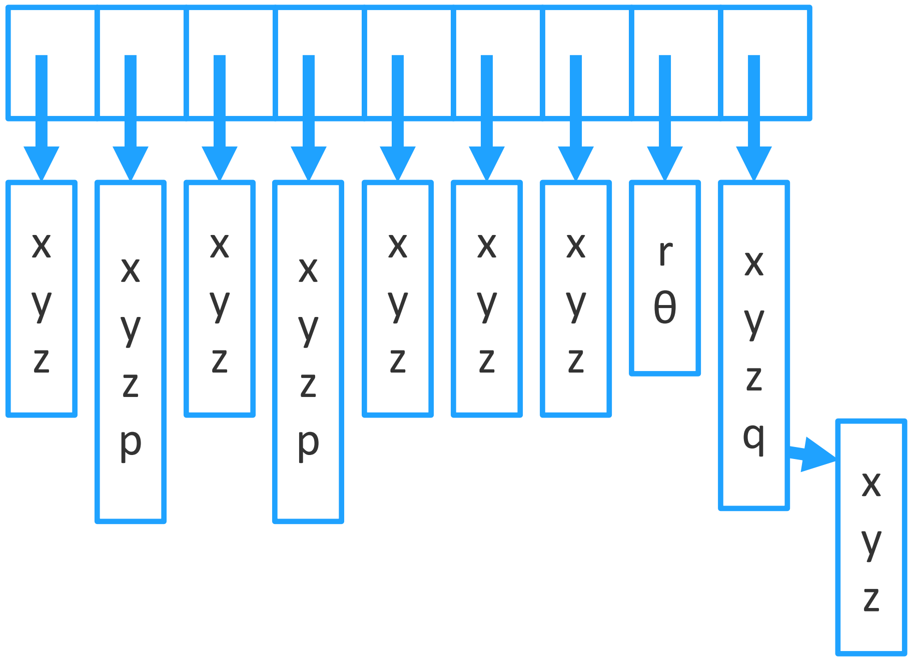
```
for (i=0; i<N; ++i) {
        points[i]->incx();
}
```



Conflicting Goals

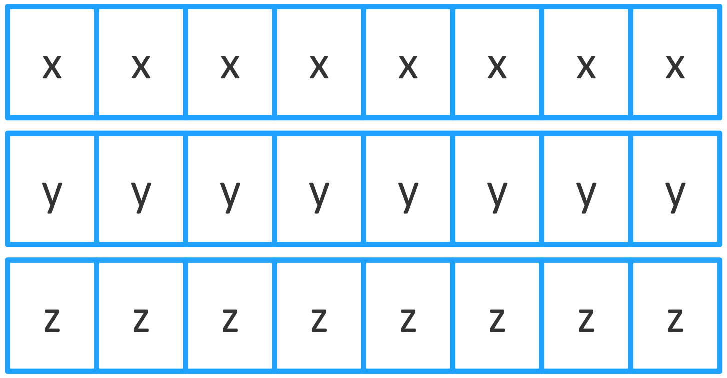
- Some well-motivated software structures have real value, but make acceleration harder
- Examples:
  - Virtual method calls inside a loop
  - Collections with non-uniform type
  - Substructure sharing

```
for (i=0; i<N; ++i) {
        points[i]->incx();
}
```



What makes Acceleration easier?

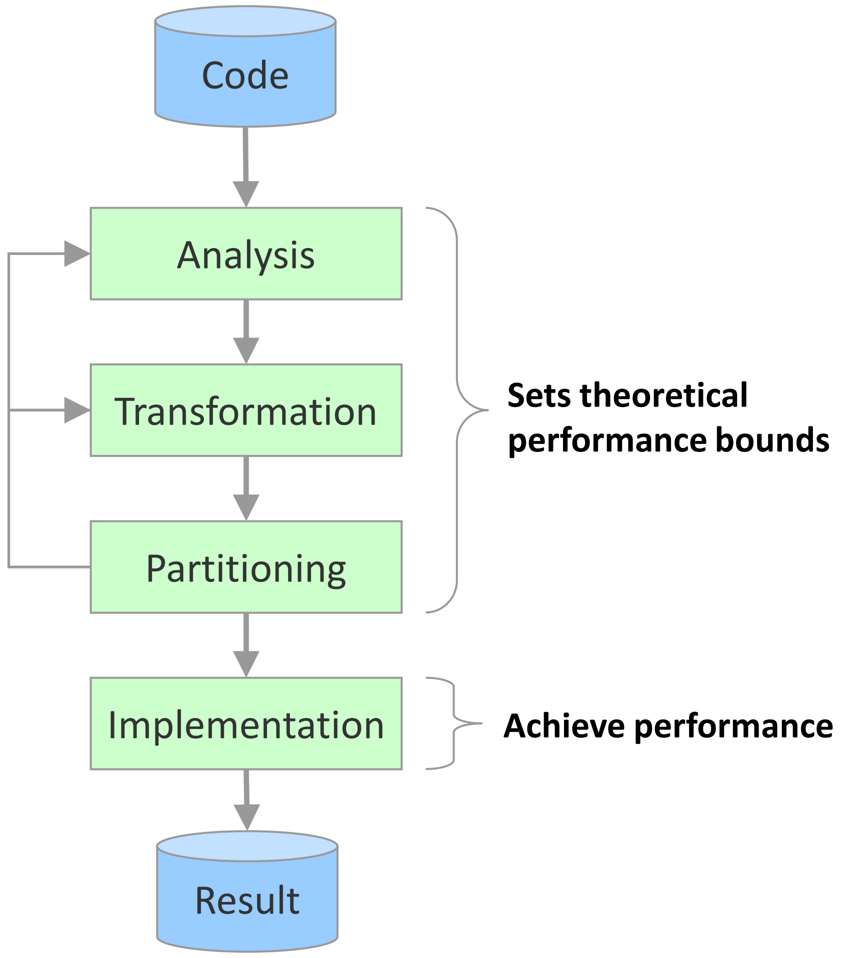
- Self-evident data dependences
- Computing on large collections of uniform data
- Appropriate representation hiding
- Getting the abstraction right



Maximum Performance Computing

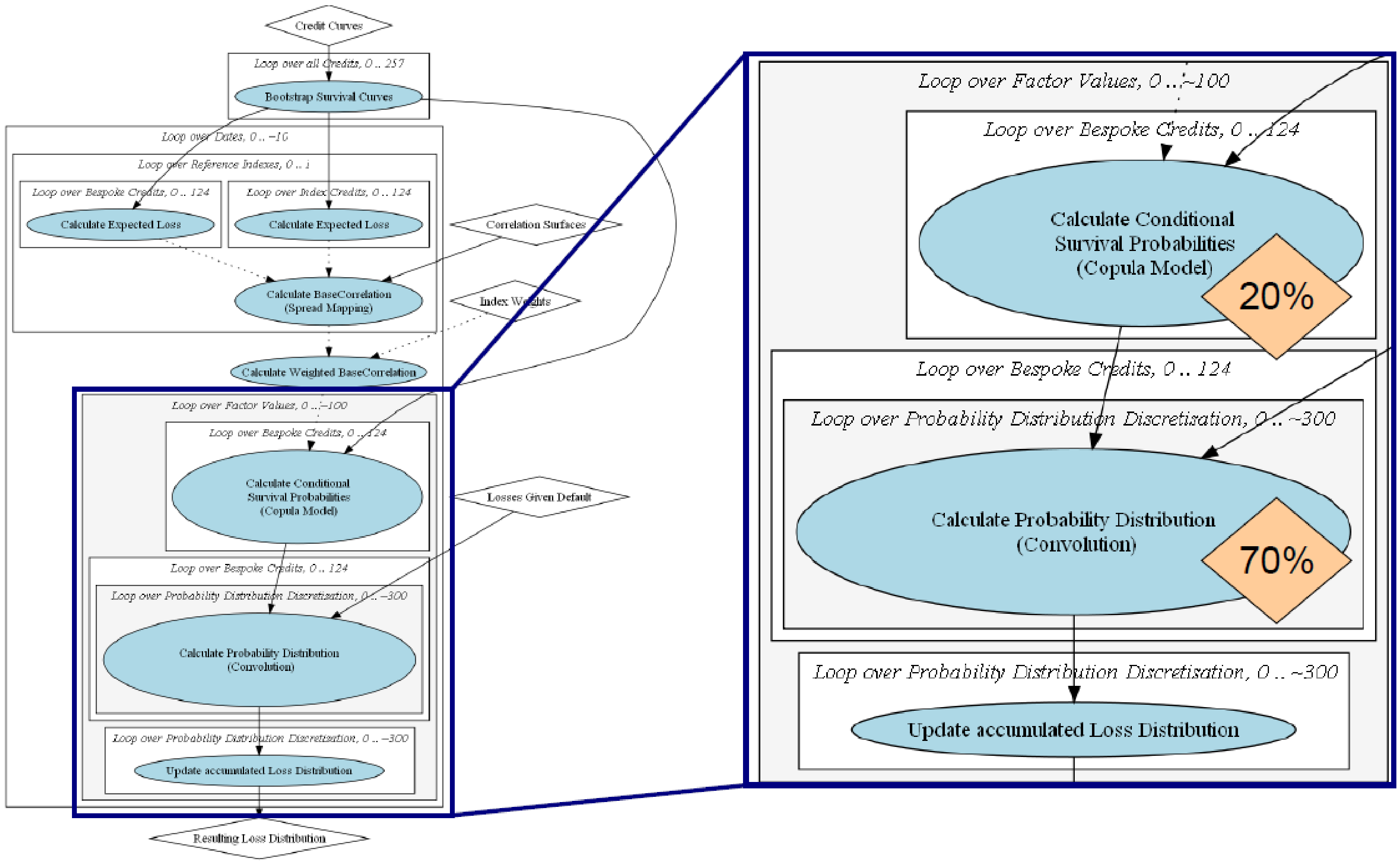
- Identify parallelism and take advantage of it
  - Fully understand data dependencies
- Minimize memory bandwidth
  - Data reuse and representation
- Regularize the computation and data
  - Minimize control flow complexity
- Find optimal balance for underlying architecture
  - Memory hierarchy bandwidth(s) and size(s) and latency(s)
  - Communication bandwidth(s) and latency(s)
  - Math performance
  - Branch cost (control divergence)
  - Axes of Parallelism

Maxeler Acceleration Process



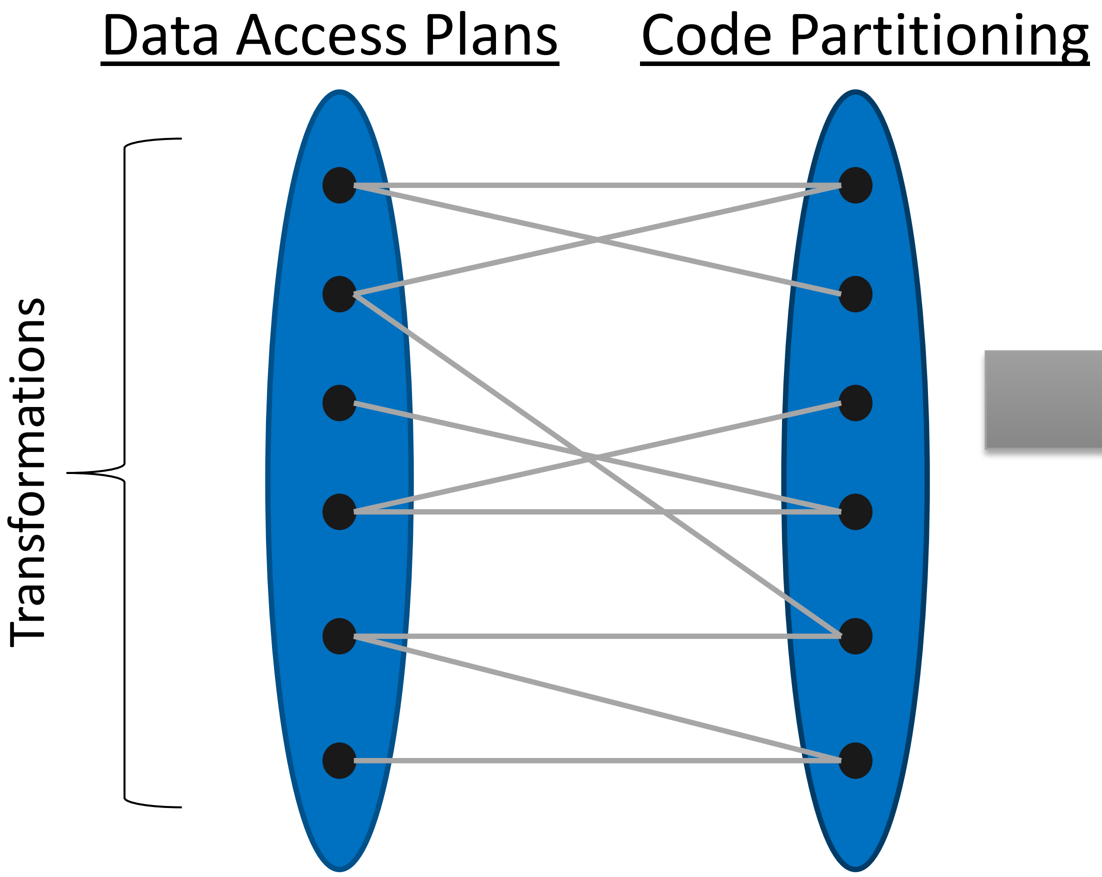
- Run the code with profiling tools
- Understand data and loop structures and data access patterns
- Investigate transformation options for these structures and access patterns
- Decide which parts of the code need acceleration
- Implement and validate

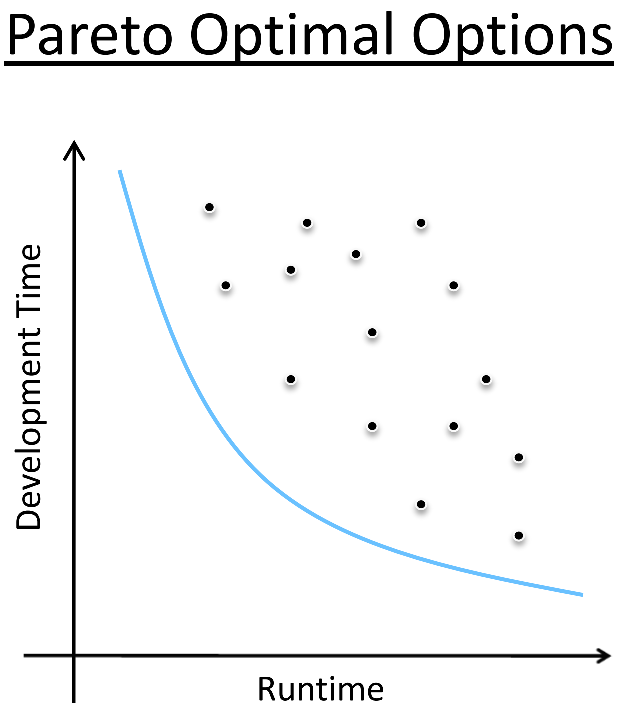
Application Analysis



Credit Curves
Loop over all Credits, 0 .. 257
Bootstrap Survival Curves
Loop over Dates, 0 .. ~10
Loop over Reference Indexes, 0 .. i
Loop over Bespoke Credits, 0 .. 124
Calculate Expected Loss
Loop over Index Credits, 0 .. 124
Calculate Expected Loss
Correlation Surfaces
Calculate BaseCorrelation (Spread Mapping)
Index Weights
Calculate Weighted BaseCorrelation
Loop over Factor Values, 0 .. ~100
Loop over Bespoke Credits, 0 .. 124
Calculate Conditional Survival Probabilities (Copula Model)
Losses Given Default
Loop over Bespoke Credits, 0 .. 124
Loop over Probability Distribution Discretisation, 0 .. ~300
Calculate Probability Distribution (Convolution)
Loop over Probability Distribution Discretisation, 0 .. ~300
Update accumulated Loss Distribution
Resulting Loss Distribution
Loop over Factor Values, 0 .. ~100
Loop over Bespoke Credits, 0 .. 124
Calculate Conditional Survival Probabilities (Copula Model)
20%
Loop over Bespoke Credits, 0 .. 124
Loop over Probability Distribution Discretisation, 0 .. ~300
Calculate Probability Distribution (Convolution)
70%
Loop over Probability Distribution Discretisation, 0 .. ~300
Update accumulated Loss Distribution

Partitioning Options

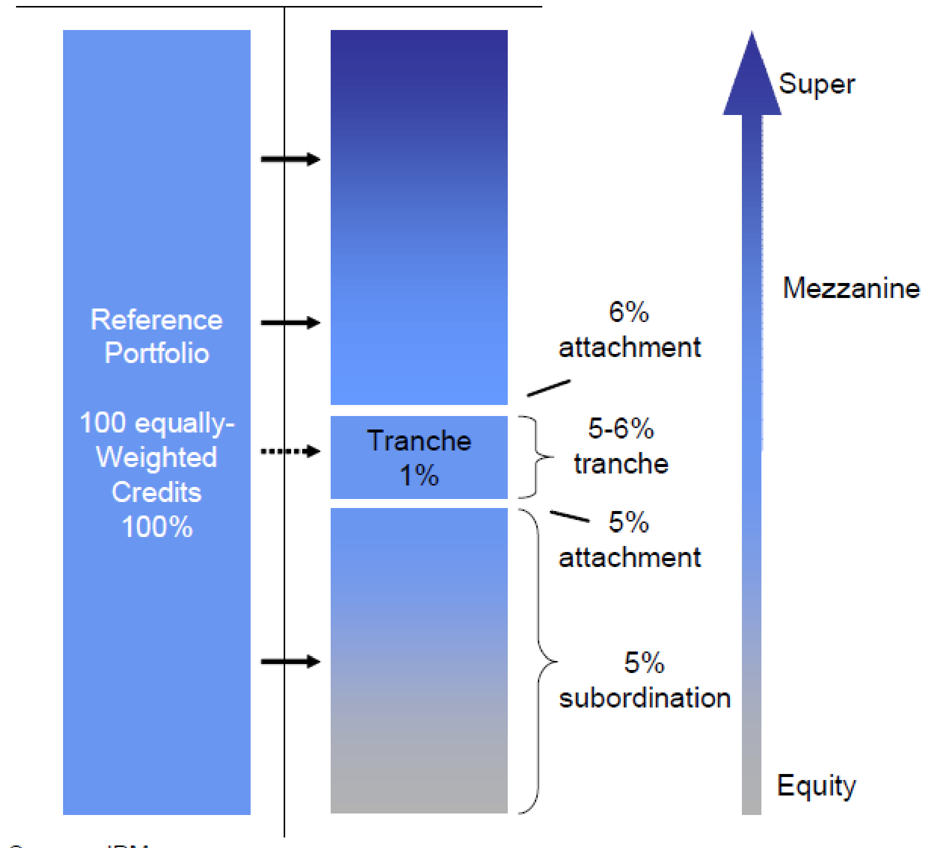




Try to minimise runtime and development time, while maximising flexibility and precision.

Credit Derivatives Valuation & Risk

- Compute value of complex financial derivatives (CDOs)
- Typically run overnight, but beneficial to compute in real-time
- Many independent jobs
- Speedup: 220-270x
- Power consumption per node drops from 250W to 235W/node



Source: JPMorgan

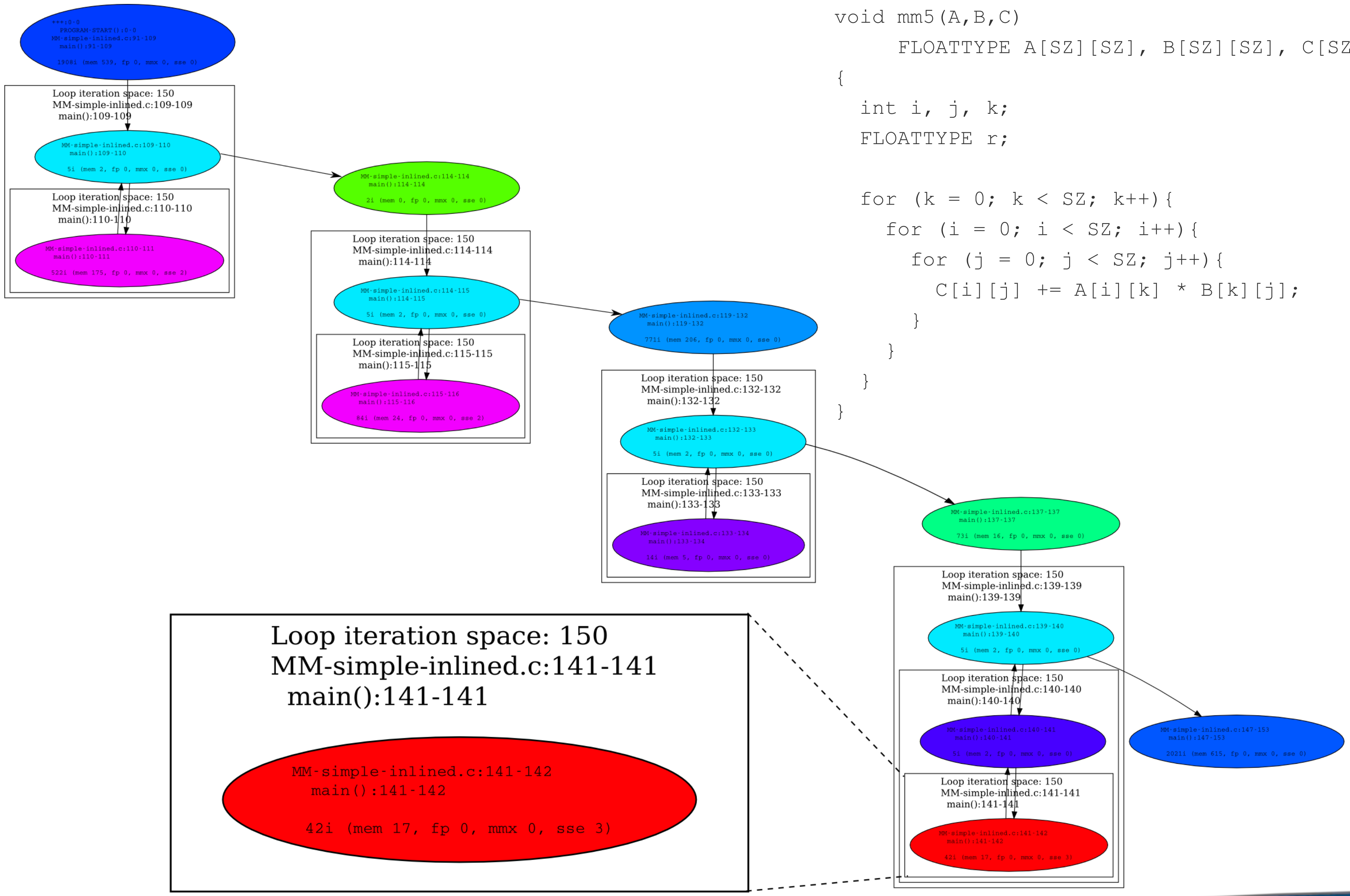
Discovering the Dataflow of an Application

MAXELER
Technologies
MAXIMUM PERFORMANCE COMPUTING

MaxSpot

- Developed in-house to make deciphering complex code easier
- MaxSpot is a tool to profile, analyse, and visualise the dynamic behaviour of applications
- Extensible analysis framework
- Determines control-flow and data-flow
- Build *loop graphs*
- Runs on application binaries
  - Independent of original programming languages(s)
  - Execute MaxSpot with one (or more) test data-sets and observe code paths

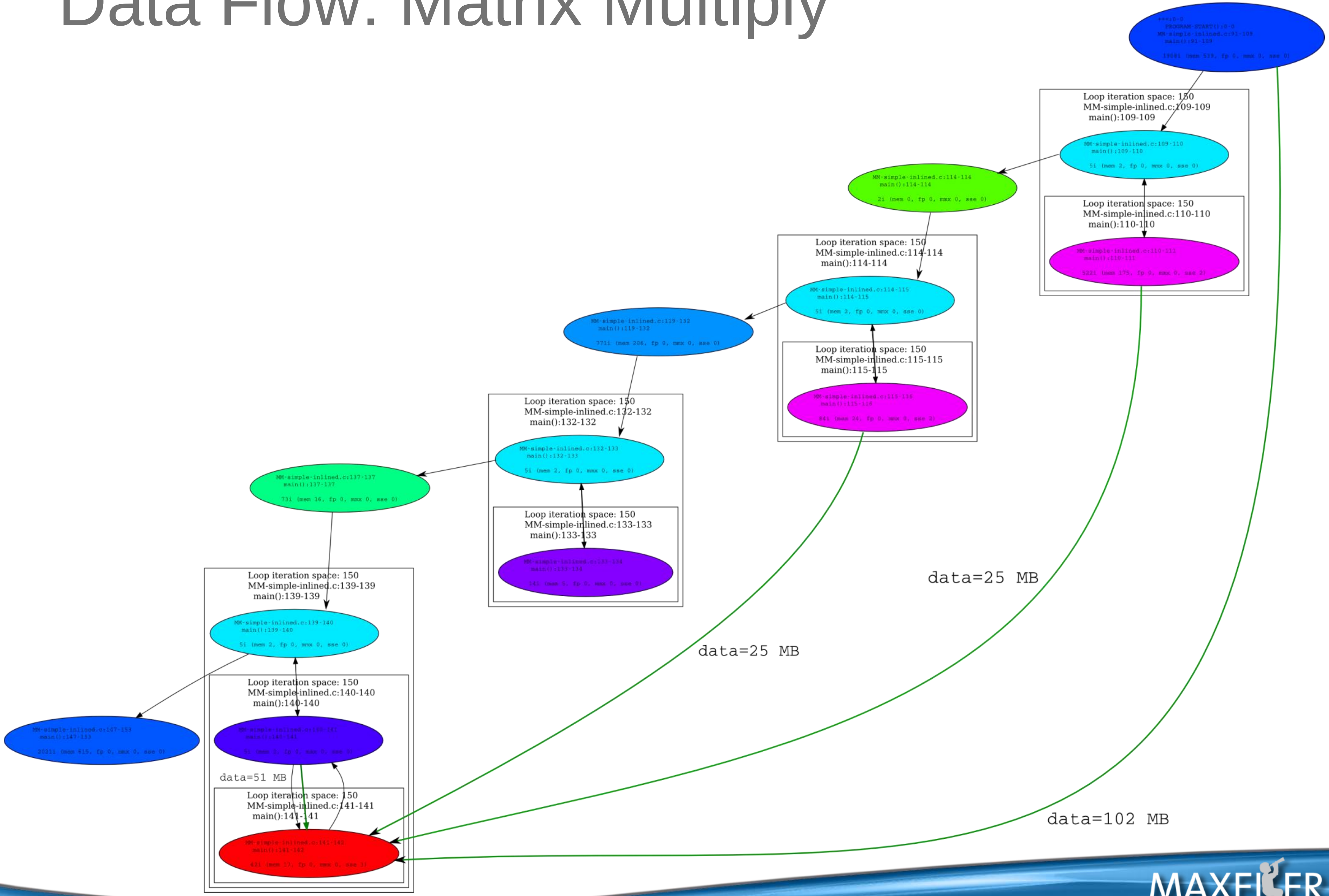
Control Flow: Matrix Multiply



+++:0-0 PROGRAM-START():0-0 MM-simple-inlined.c:91-109 main():91-109 1908i (mem 539, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:109-109 main():109-109
MM-simple-inlined.c:109-110 main():109-110 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:110-110 main():110-110
MM-simple-inlined.c:110-111 main():110-111 522i (mem 175, fp 0, mmx 0, sse 2)
MM-simple-inlined.c:114-114 main():114-114 2i (mem 0, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:114-114 main():114-114
MM-simple-inlined.c:114-115 main():114-115 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:115-115 main():115-115
MM-simple-inlined.c:115-116 main():115-116 84i (mem 24, fp 0, mmx 0, sse 2)
MM-simple-inlined.c:119-132 main():119-132 771i (mem 206, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:132-132 main():132-132
MM-simple-inlined.c:132-133 main():132-133 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:133-133 main():133-133
MM-simple-inlined.c:133-134 main():133-134 14i (mem 5, fp 0, mmx 0, sse 0)
MM-simple-inlined.c:137-137 main():137-137 73i (mem 16, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:139-139 main():139-139
MM-simple-inlined.c:139-140 main():139-140 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:140-140 main():140-140
MM-simple-inlined.c:140-141 main():140-141 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:141-141 main():141-141
MM-simple-inlined.c:141-142 main():141-142 42i (mem 17, fp 0, mmx 0, sse 3)
MM-simple-inlined.c:147-153 main():147-153 2021i (mem 615, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:141-141 main():141-141
MM-simple-inlined.c:141-142 main():141-142 42i (mem 17, fp 0, mmx 0, sse 3)

```
void mm5(A,B,C)
     FLOATTYPE A[SZ][SZ], B[SZ][SZ], C[SZ][SZ];
{
  int i, j, k;
  FLOATTYPE r;

  for (k = 0; k < SZ; k++){
    for (i = 0; i < SZ; i++){
      for (j = 0; j < SZ; j++){
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

Data Flow: Matrix Multiply



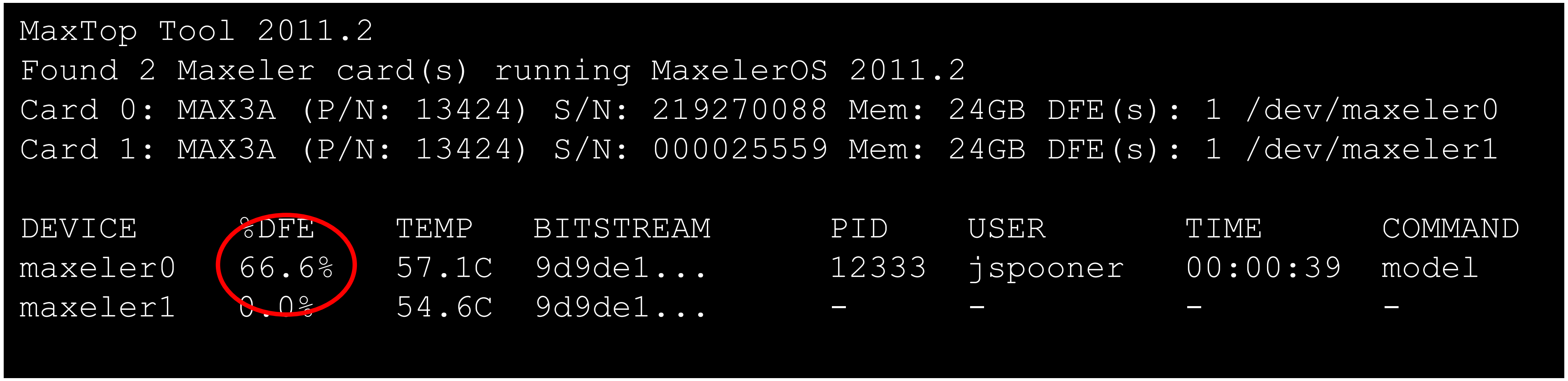
+++:0-0 PROGRAM-START():0-0 MM-simple-inlined.c:91-109 main():91-109 19081 (mem 539, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:109-109 main():109-109
MM-simple-inlined.c:109-110 main():109-110 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:110-110 main():110-110
MM-simple-inlined.c:110-111 main():110-111 522i (mem 175, fp 0, mmx 0, sse 2)
MM-simple-inlined.c:114-114 main():114-114 2i (mem 0, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:114-114 main():114-114
MM-simple-inlined.c:114-115 main():114-115 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:115-115 main():115-115
MM-simple-inlined.c:115-116 main():115-116 84i (mem 24, fp 0, mmx 0, sse 2)
MM-simple-inlined.c:119-132 main():119-132 771i (mem 206, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:132-132 main():132-132
MM-simple-inlined.c:132-133 main():132-133 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:133-133 main():133-133
MM-simple-inlined.c:133-134 main():133-134 14i (mem 5, fp 0, mmx 0, sse 0)
MM-simple-inlined.c:137-137 main():137-137 73i (mem 16, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:139-139 main():139-139
MM-simple-inlined.c:139-140 main():139-140 5i (mem 2, fp 0, mmx 0, sse 0)
Loop iteration space: 150 MM-simple-inlined.c:140-140 main():140-140
MM-simple-inlined.c:140-141 main():140-141 5i (mem 2, fp 0, mmx 0, sse 0)
MM-simple-inlined.c:147-153 main():147-153 2021i (mem 615, fp 0, mmx 0, sse 0)
data=51 MB
Loop iteration space: 150 MM-simple-inlined.c:141-141 main():141-141
MM-simple-inlined.c:141-142 main():141-142 42i (mem 17, fp 0, mmx 0, sse 3)
data=25 MB
data=25 MB
data=102 MB

Performance and Profiling of Accelerated Systems

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Measuring Utilization

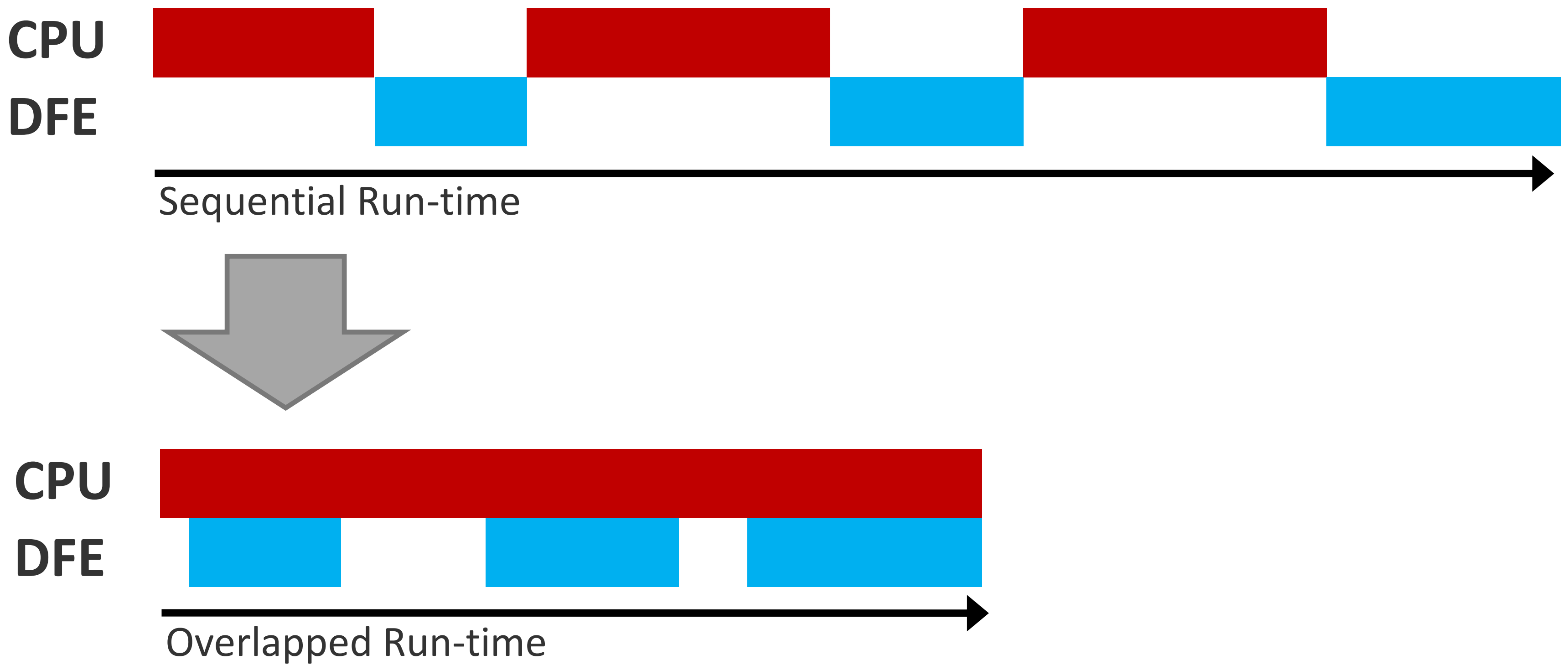
- *Top* measures % of time CPU is running
- *Maxtop* monitors % of time the DFE is running



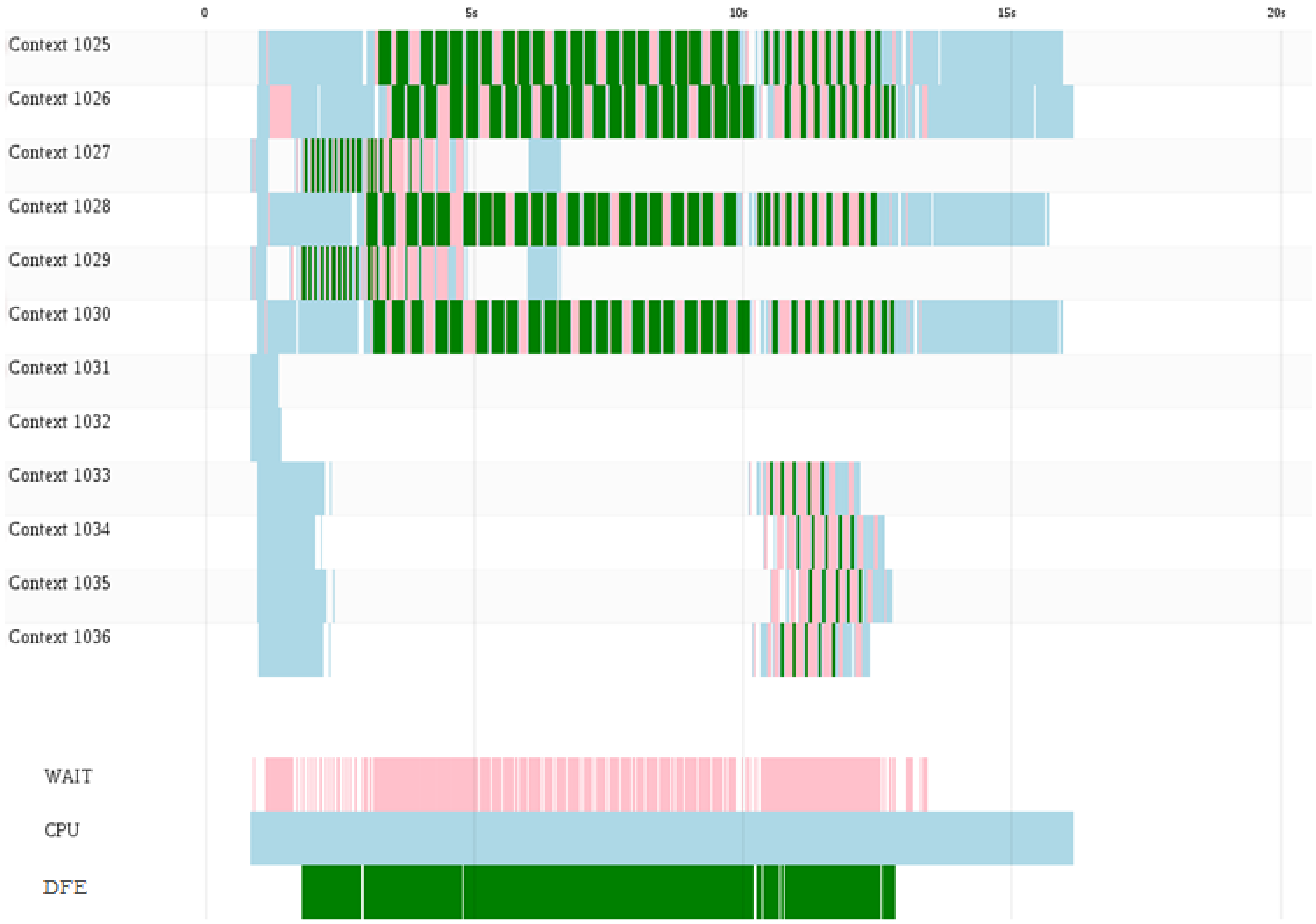
```
MaxTop Tool 2011.2
Found 2 Maxeler card(s) running MaxelerOS 2011.2
Card 0: MAX3A (P/N: 13424) S/N: 219270088 Mem: 24GB DFE(s): 1 /dev/maxeler0
Card 1: MAX3A (P/N: 13424) S/N: 000025559 Mem: 24GB DFE(s): 1 /dev/maxeler1

DEVICE      %DFE    TEMP   BITSTREAM      PID    USER       TIME       COMMAND
maxeler0    66.6%   57.1C  9d9de1...      12333  jspooner   00:00:39   model
maxeler1    0.0%    54.6C  9d9de1...      -      -          -          -
```

Overlapping CPU + DFE

- CPU and DFE can (and should!) process in parallel
  - Runtime always limited by longest running part



Performance Profiling



MAXELER Technologies

Maxeler University Program Members

KOREA UNIVERSITY

UNIVERSITÄT PADERBORN

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

NANYANG TECHNOLOGICAL UNIVERSITY

STANFORD UNIVERSITY

NUS National University of Singapore

POLITECNICO DI MILANO

THE UNIVERSITY OF TOKYO

東北大学 TOHOKU UNIVERSITY

復旦大學

清華大學 Tsinghua University

Manchester Metropolitan University

TECHNION Israel Institute of Technology

Imperial College London

JAIST JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY 1990

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

UH Universität Hamburg DER FORSCHUNG | DER LEHRE | DER BILDUNG

UiO University of Oslo

Conclusions

- The challenge is to make the best use of Silicon we can
- Frequency Scaling is over, it’s time to start thinking in parallel
- Heterogeneous system design allows us to tailor systems to the applications
- Ultra-fine-grained parallelism in Dataflow computing benefits throughput and latency