第一讲：
Data Flow Computing

付昊桓

haohuan@tsinghua.edu.cn

付昊桓：个人简介

1999 – 2003:
Tsinghua University, B.E.
Computer Science

2010 – now:
Tsinghua University, Associate Professor
High Performance Geo-Computing

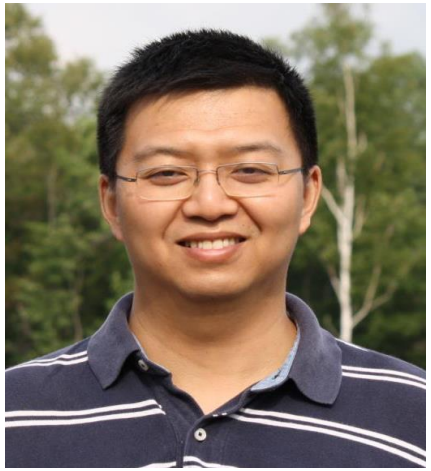
2009 – 2010:
Stanford University, PostDoc
HPC in Geophysics

A

B

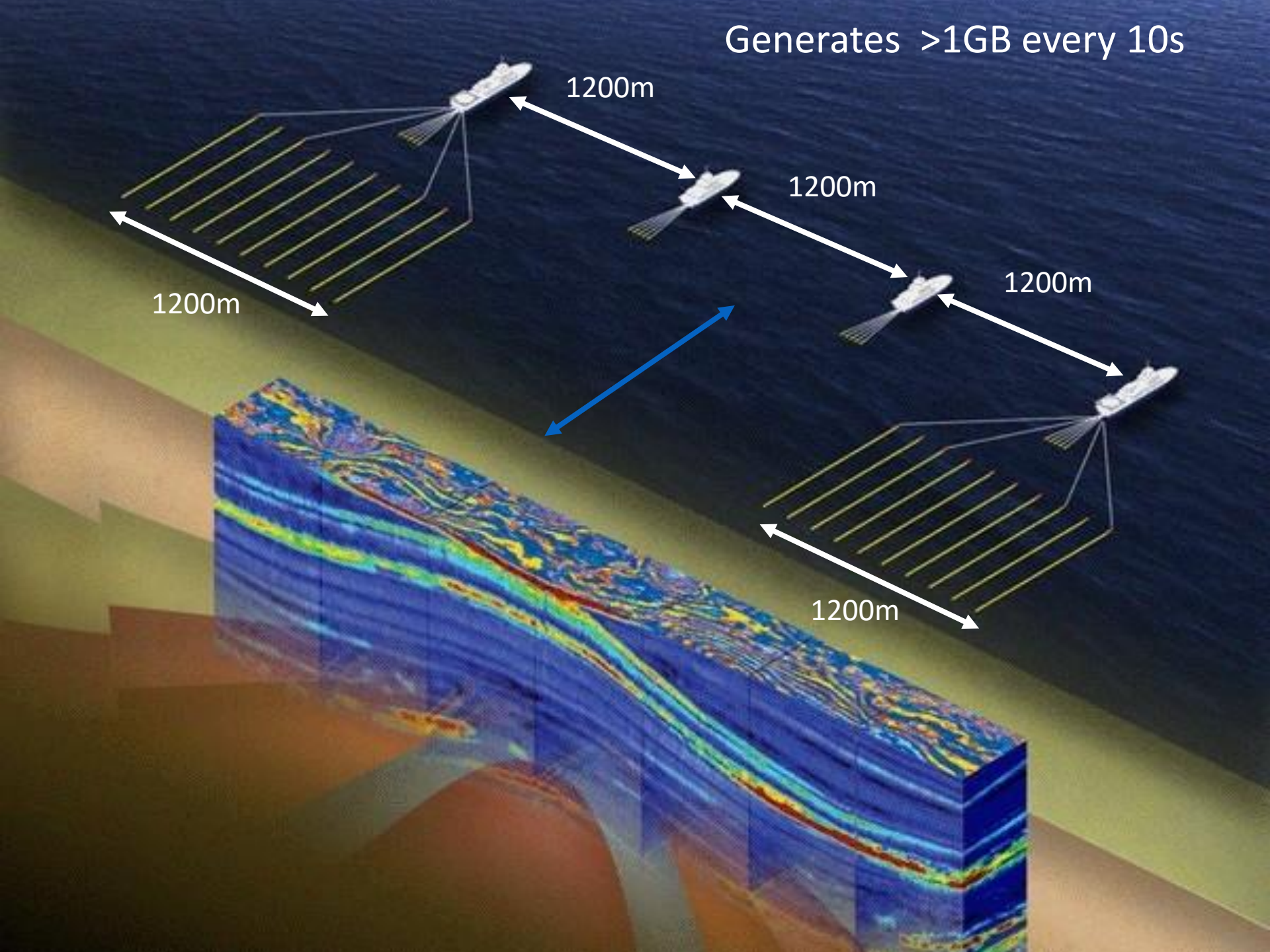
2003 – 2005:
City Univeresity of Hong Kong, MPhil
Computer Science (wireless network)

2005 – 2009:
Imperial College London, PhD
Computing (reconfigurable computing)

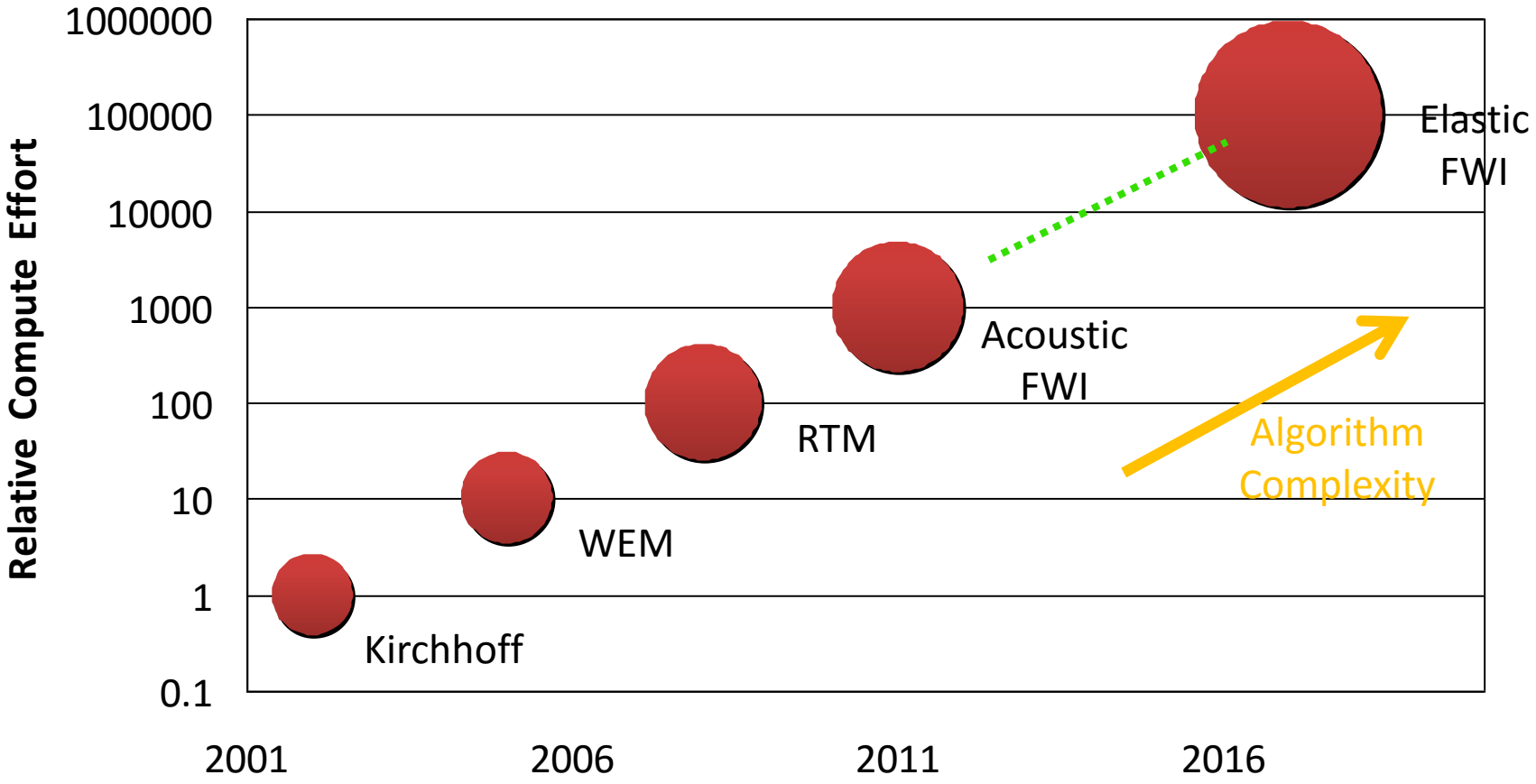


高性能计算所面临的挑战

- 应用发展趋势
  - 大数据（Big Data）
  - 模型越来越精细，算法越来越复杂


Generates >1GB every 10s
1200m
1200m
1200m
1200m
1200m

Computational Challenges: Complexity

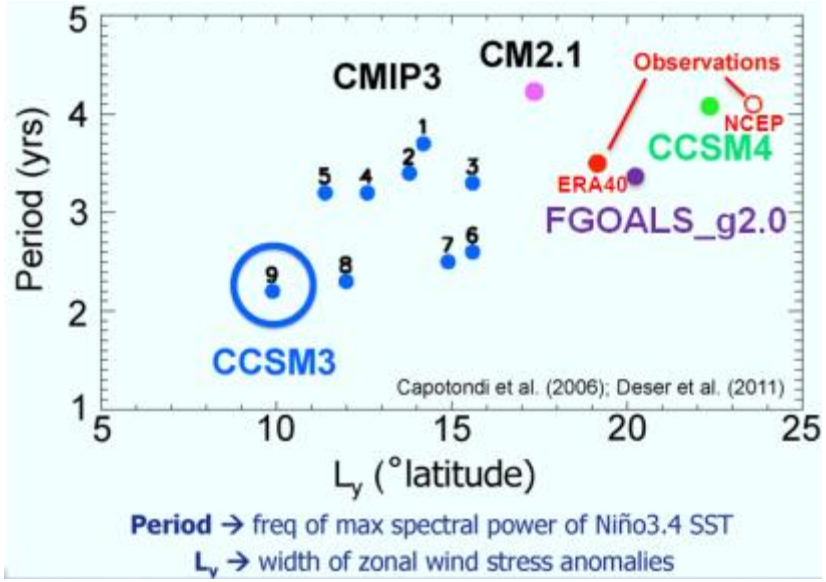


IPCC AR5 全球气候模拟

- 通过调试大气模式GAMIL、海冰模式CICE_lasg_v1.0与耦合器CPL6，解决了耦合气候系统模式FGOALS_G2.0串并行结果不一致问题
- 实现了大气分量模式GAMIL2.0经纬向二维并行剖分和MPI+OpenMP混合并行，采用240CPU核时（20个节点）GAMIL1x1的并行效率可达到40%，改进后的版本已用于IPCC AR5的相关试验。

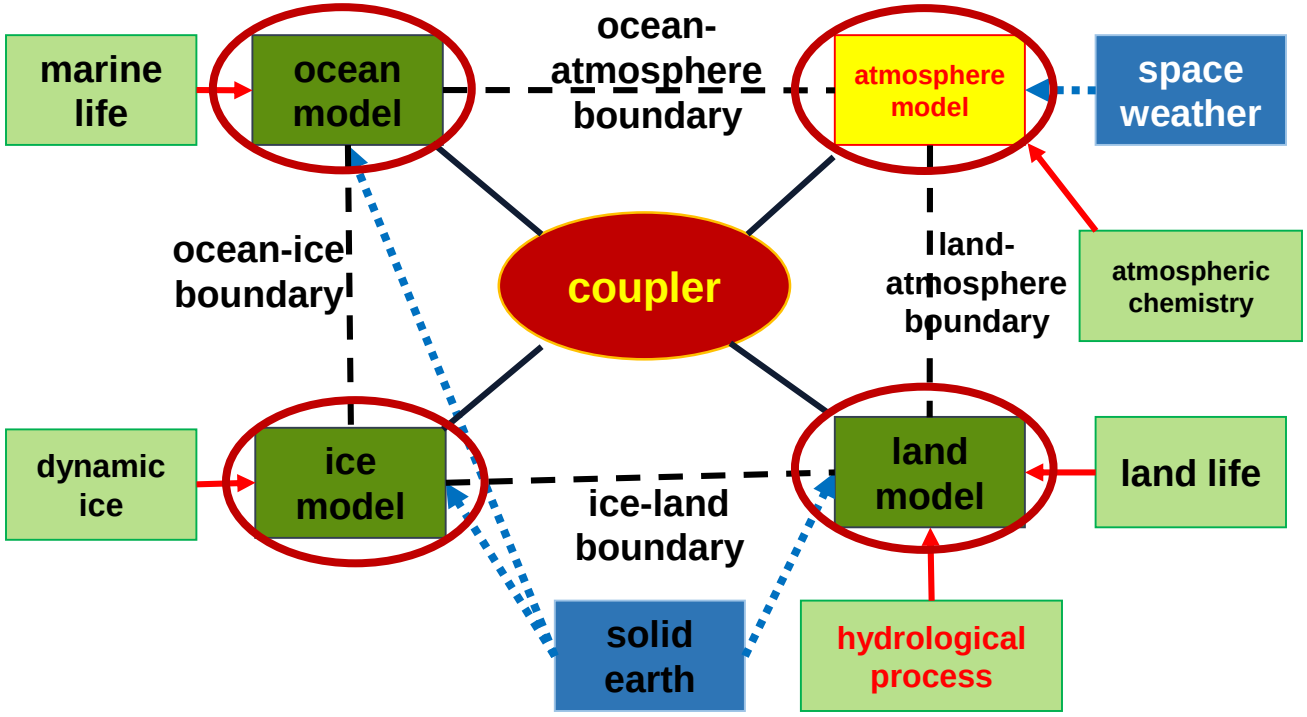
- 成功完成国际耦合模式比较计划(CMIP5)的主要模拟实验，在高性能计算平台上全系统运行4个多月，产生数据100TB

Experiments	Time length	Output data
Control Experiment	800 yrs	5.6 TB
20th Century Climate	156 yrs × 3	16.7 TB
RCP Projections	95 yrs×2+195 yrs	14.2 TB
1pCO2 and 4CO2	140+150 yrs	6.4 TB
Decadal Predictions	62 yrs × 4	8.4 TB
AMIP and CFMIP	30 yrs×7 + 5 yrs×3	27.8 TB
High Resolution Model	30 yrs	7.0 TB
CMOR outputs	(Post-processing)	11.4 TB
Total		97.5 TB



Demand for More Computing Power

- Increase in model complexity and resolution



Scale and complexity in JPMC

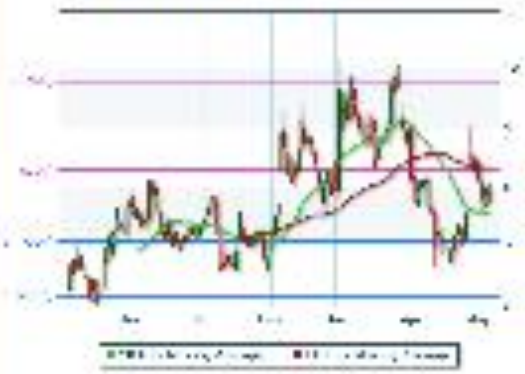
应用示例：金融

Tele-presence to 5,000 branches

Real-time fraud detection to protect our 130m card customers, billions of transactions annually

Scale & Complexity

Low-latency/high frequency trading & risk management for $Tn of trades



Optimising data transformation, storage and transmission for 160Pb of data

Need to accelerate compute to improve combination of data and reasoning

High Performance Computing

高性能计算所面临的挑战

- 高性能计算机：发展趋势
  - 多核
  - 异构
    - GPGPU
    - 众核（Intel MIC）
    - 可重构平台（FPGA）

高性能计算所面临的挑战

- 高性能计算机：问题
  - 内存带宽性能瓶颈
  - 能耗
  - 编程困难

本讲提纲

- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限

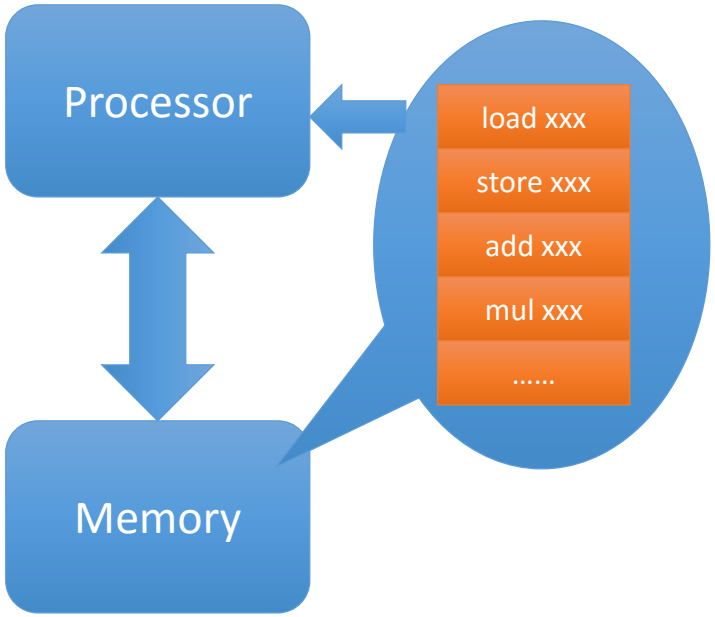
本讲提纲

- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限

Computing in Time or in Space?

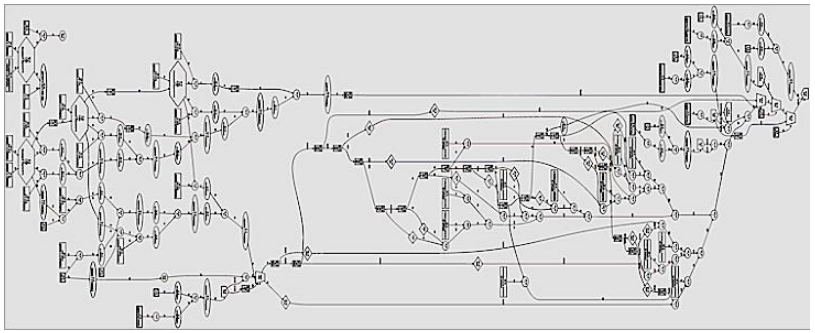
Compute in Time

- write a program that describes operations to happen in different cycles

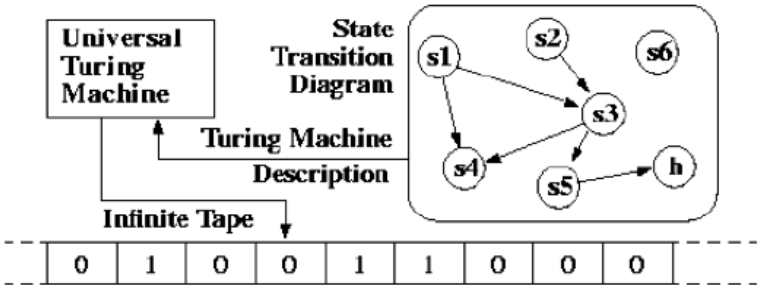


Compute in Space

- write a program that describes a specific circuit for your algorithm



传统的通用计算模式Compute in Time：简单但低效

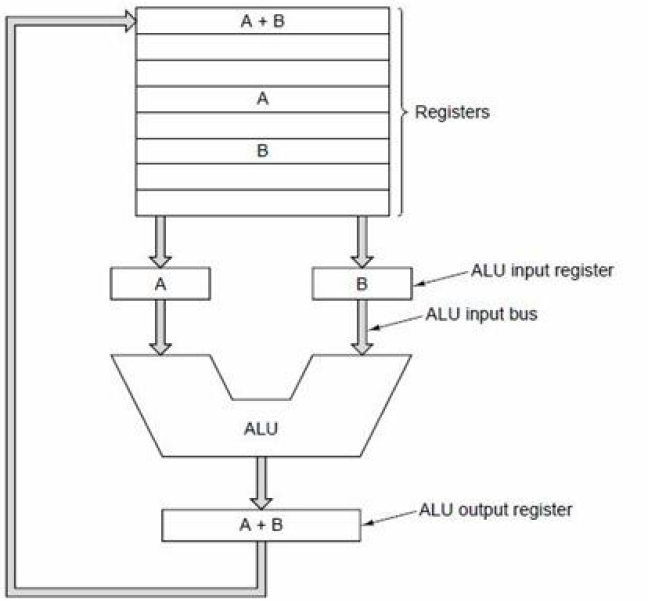


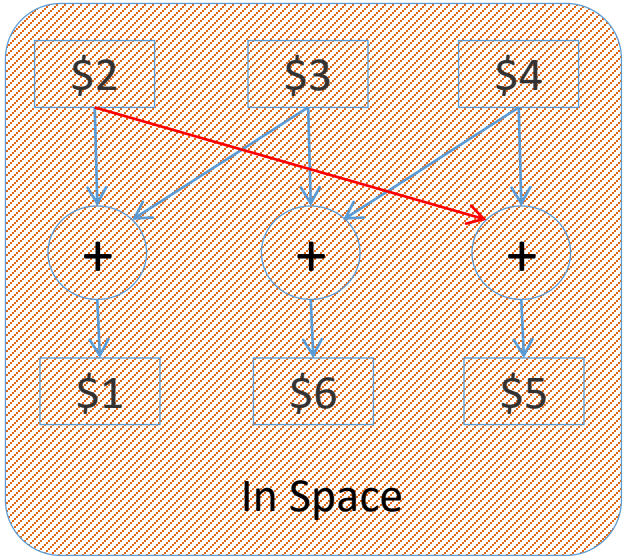
- 支持通用的计算模式
- 需要复杂的控制机制
- 不支持高效的并行执行

以在MIPS架构上执行的三个加法操作为例：

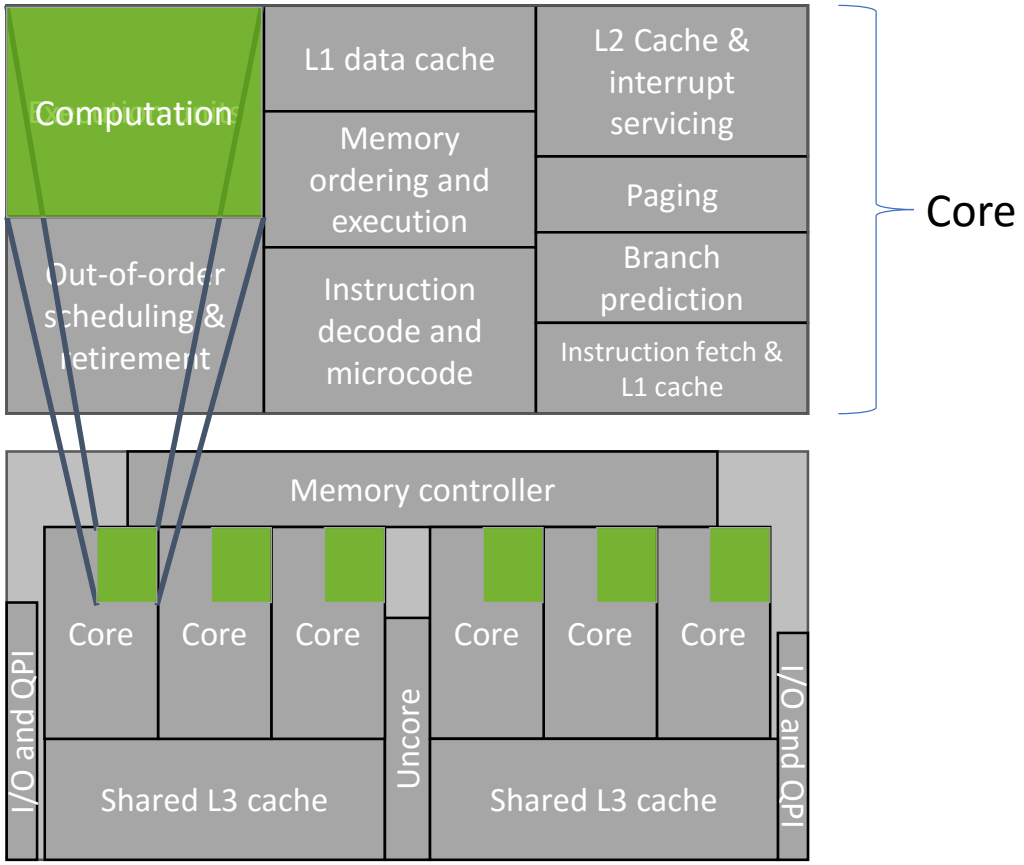
```
…
add $1,$2,$3
add $5,$4,$2
add $6,$4,$3
…
```

In Time

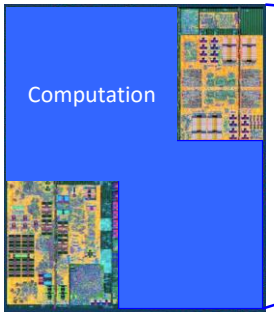




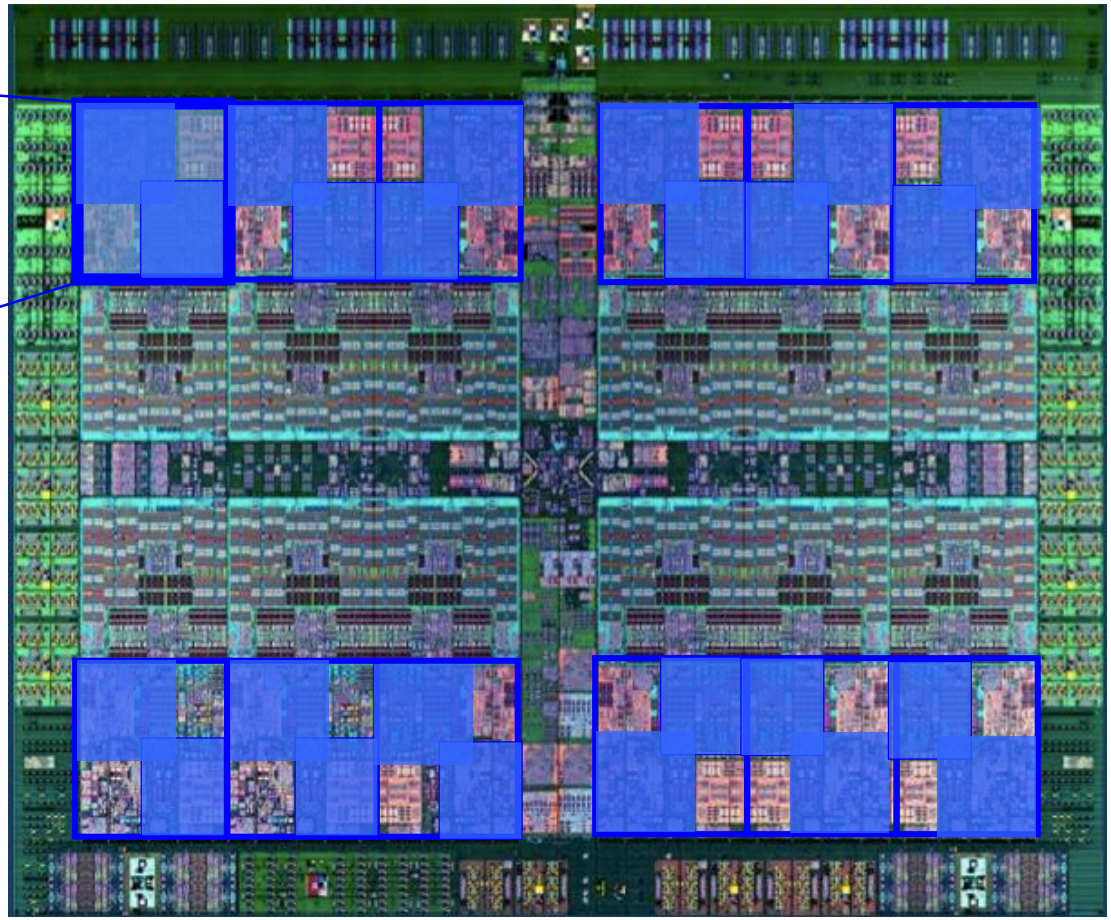
通用处理器架构: 控制和数据缓存成为“大户”
以Intel 6-Core X5680 “Westmere”为例



通用处理器架构: 控制和数据缓存成为“大户”
以IBM Power8 12-core CPU为例



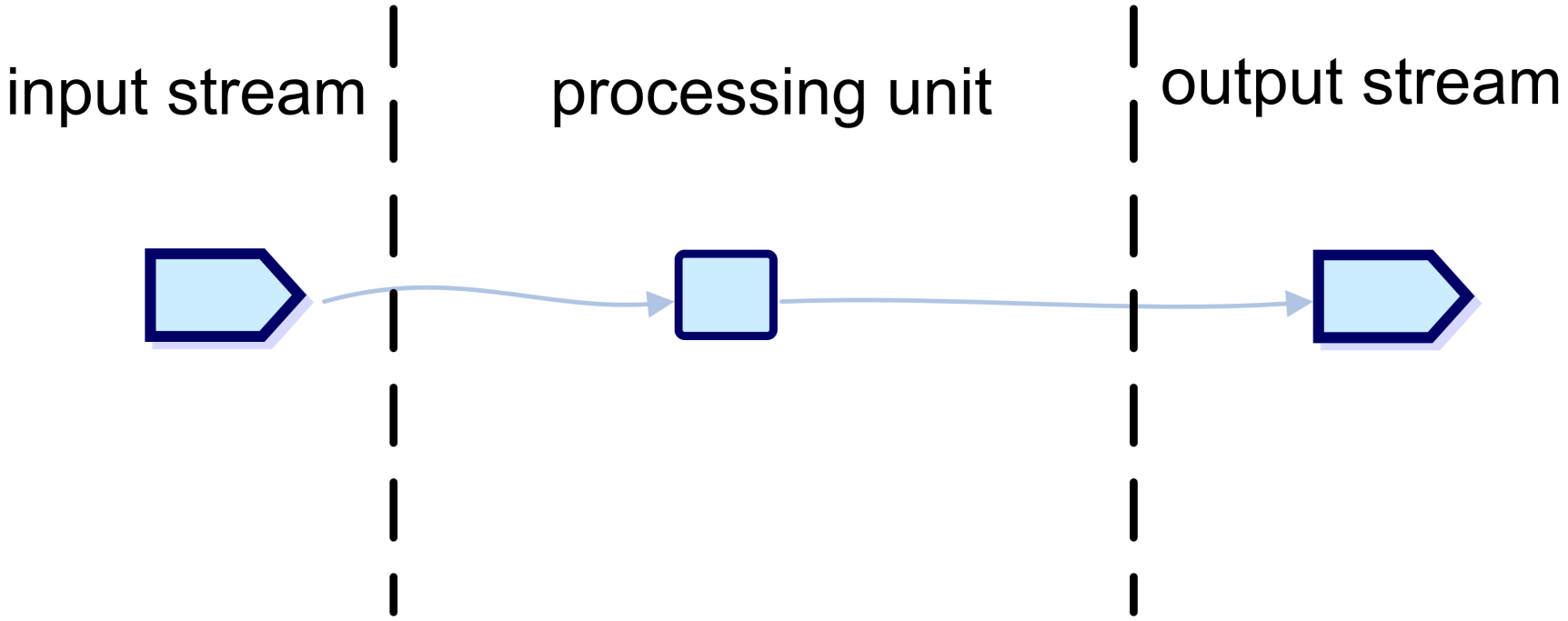
Core



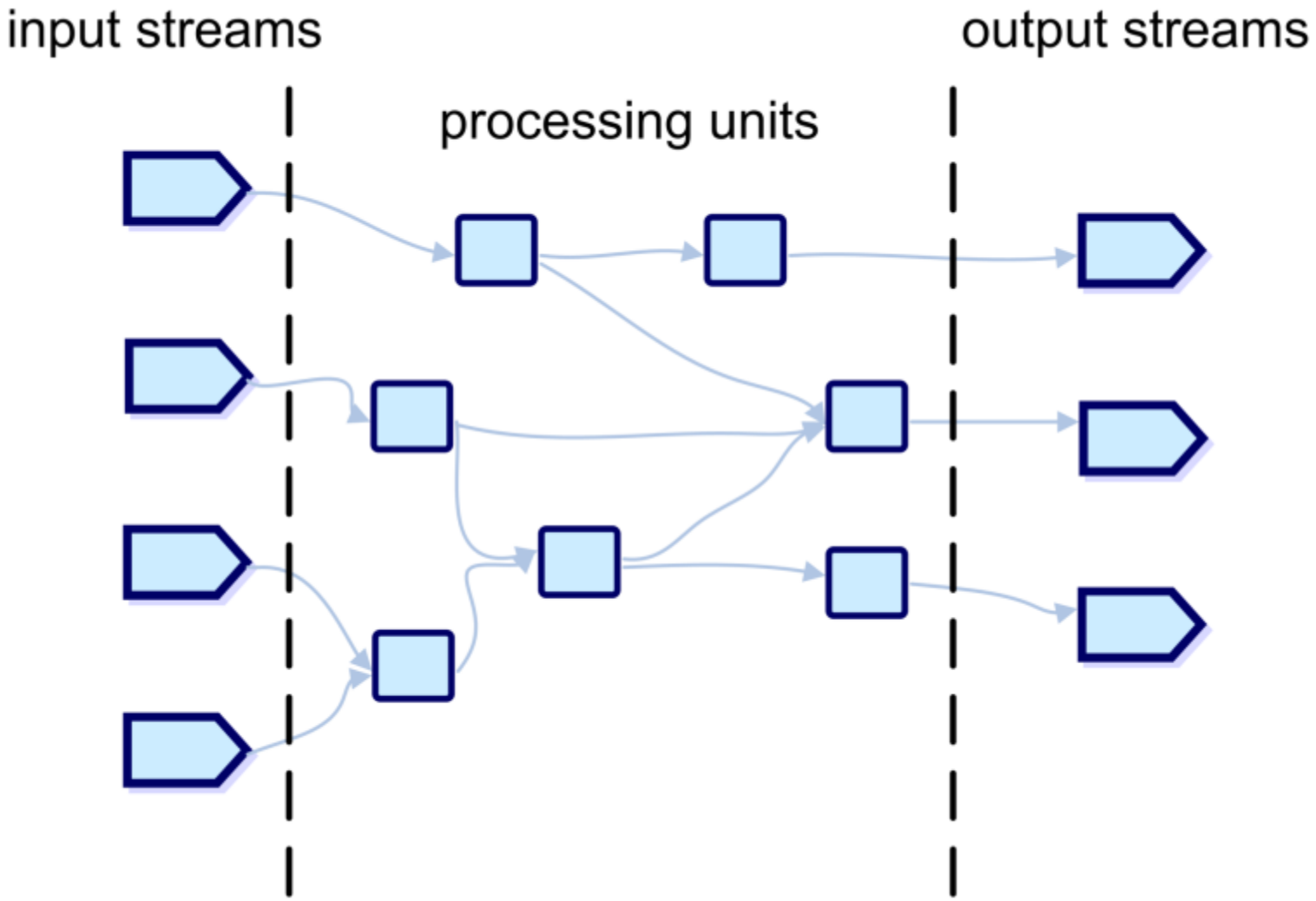
22nm SOI, eDRAM, 15 ML 650mm2,12 cores (SMT8)

Data Flow Computing: 基本概念

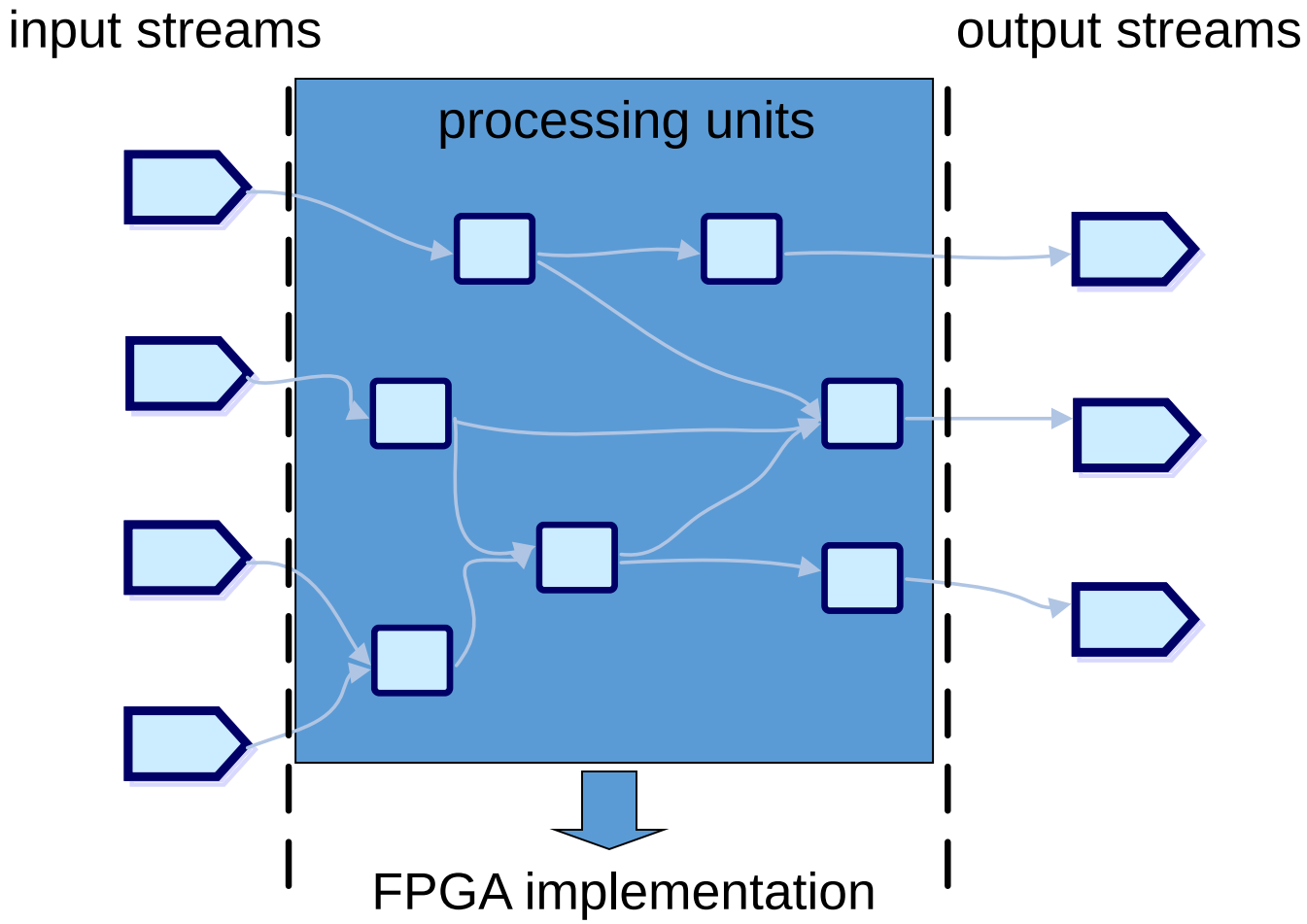
- 通过让数据流过对应的处理器件来完成计算任务。



Data Flow Computing: 基本示例

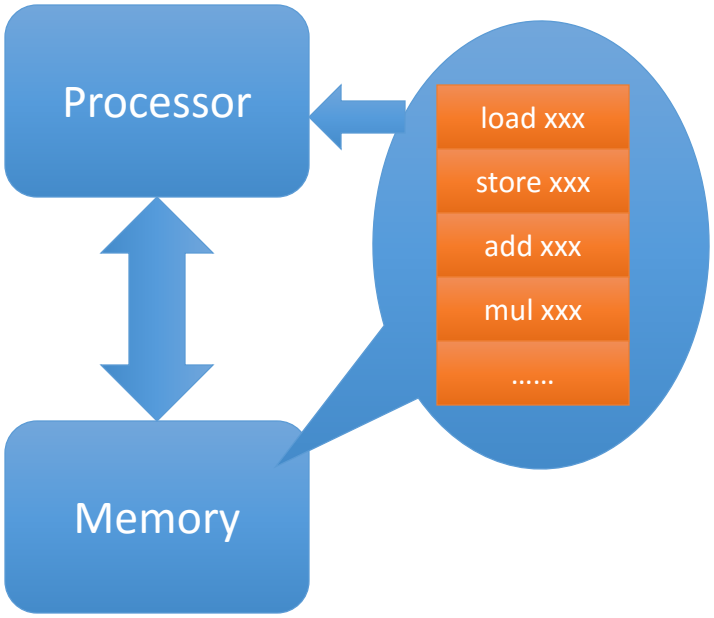


Data Flow Computing: 如何实现?

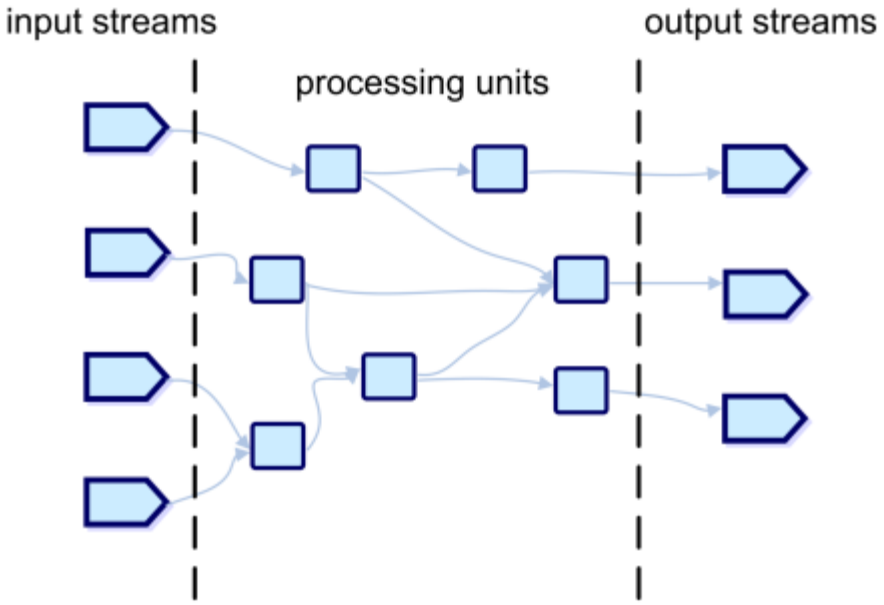


控制流计算 V.S. 数据流计算

以控制为中心

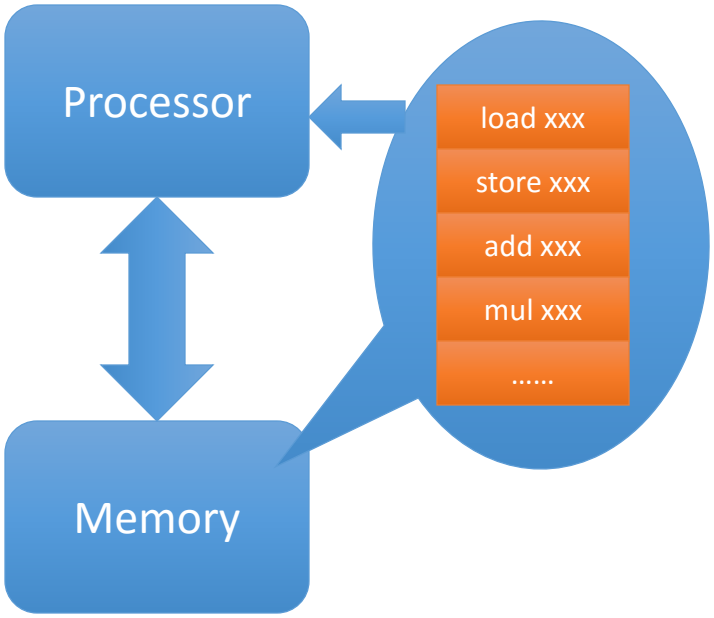


以数据流为中心

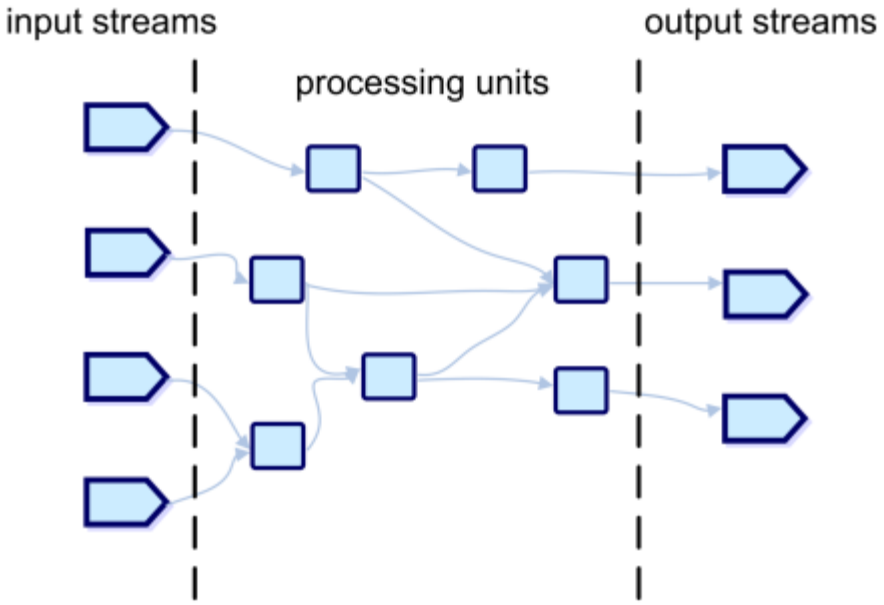


控制流计算 V.S. 数据流计算

全能型技工

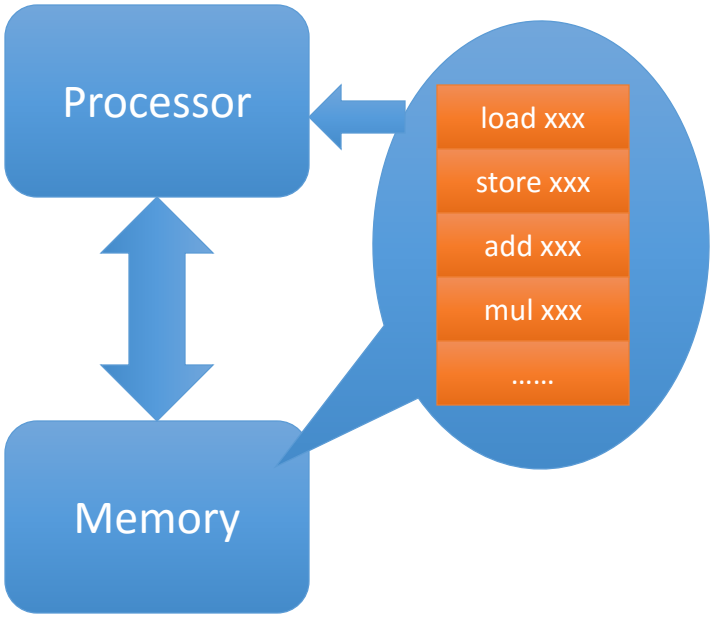


流水线工厂

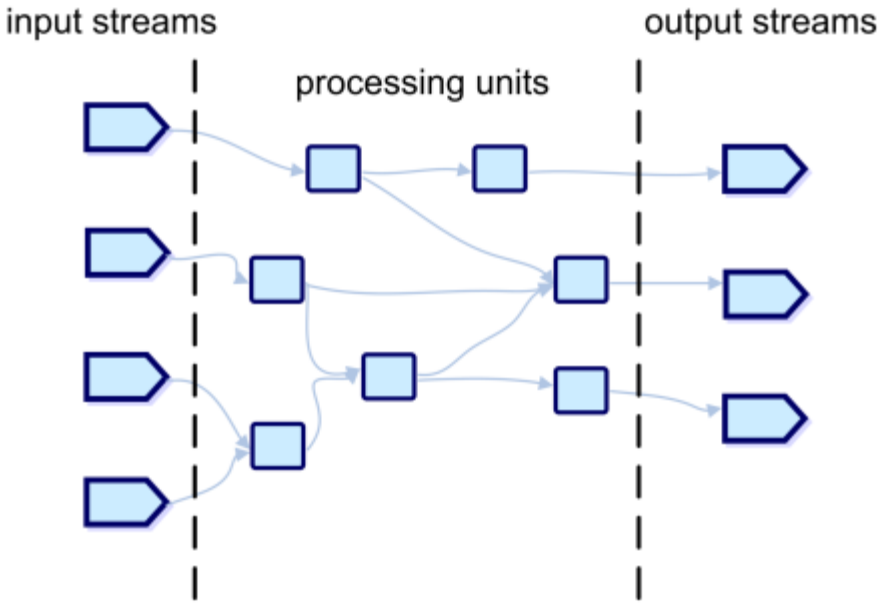


控制流计算 V.S. 数据流计算

全能型技工：全面

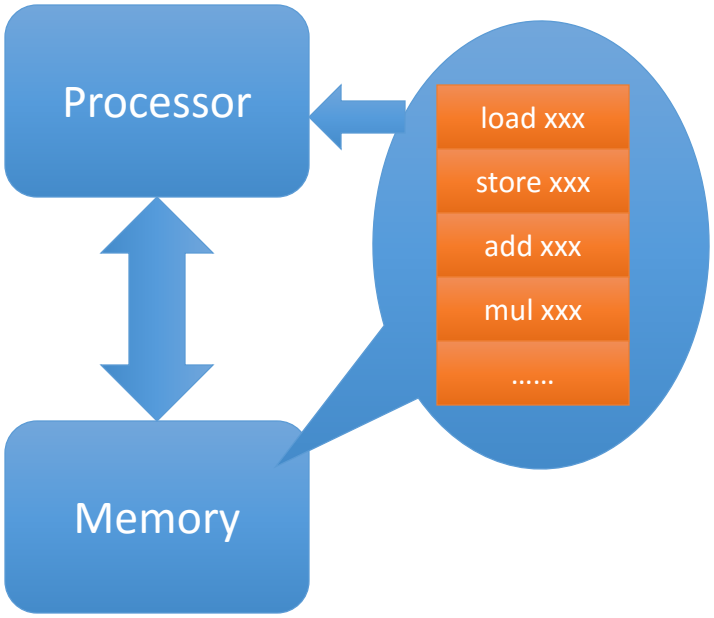


流水线工厂：专项功能

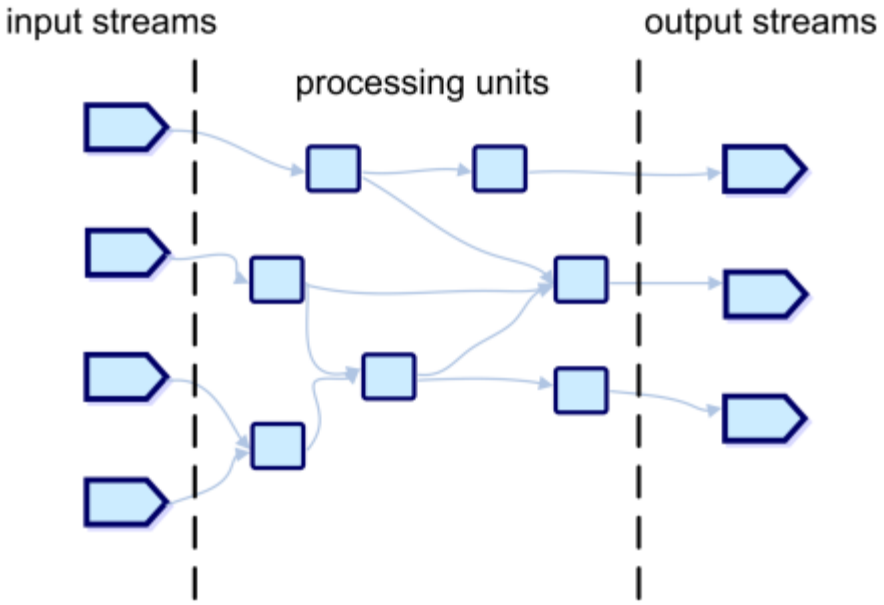


控制流计算 V.S. 数据流计算

全能型技工：低效

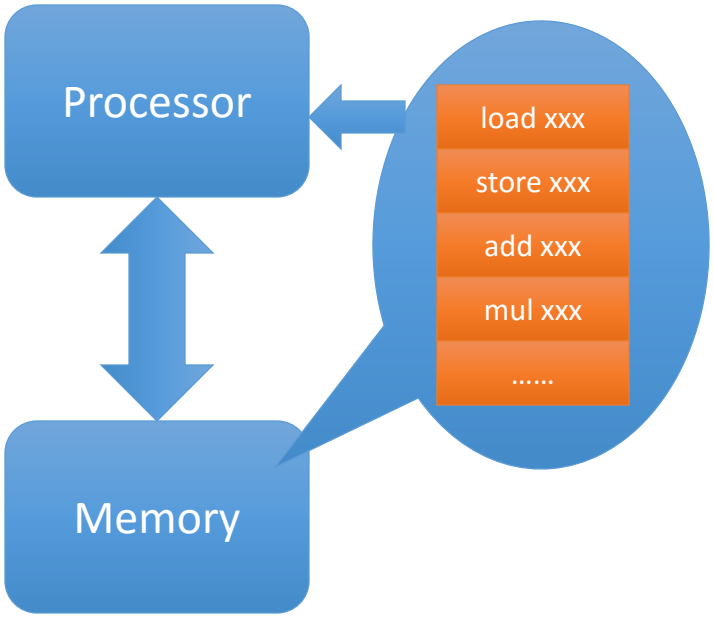


流水线工厂：高效

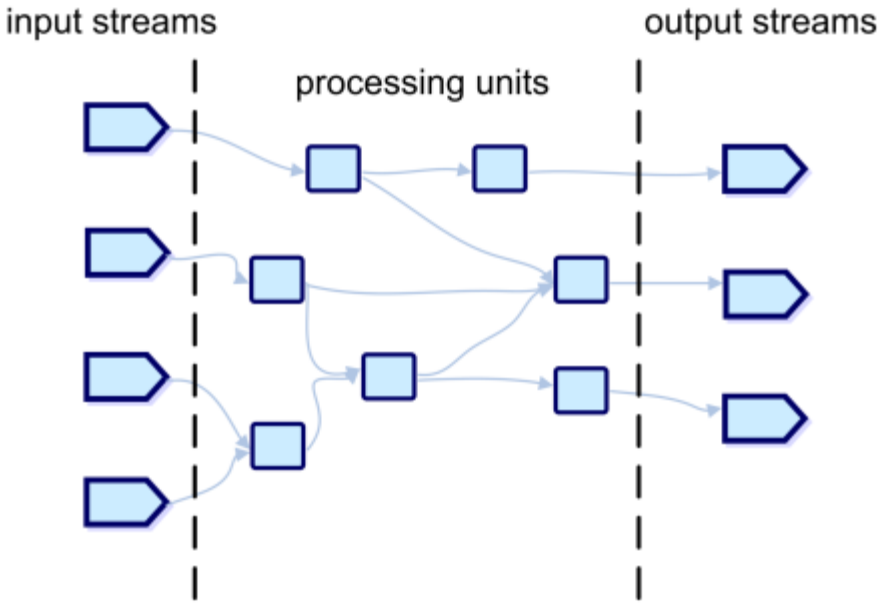


控制流计算 V.S. 数据流计算

数据：反复的读写

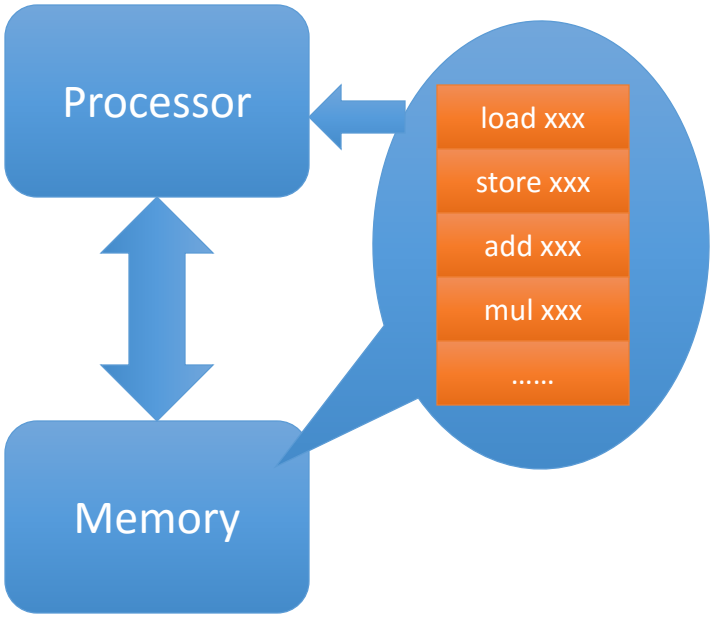


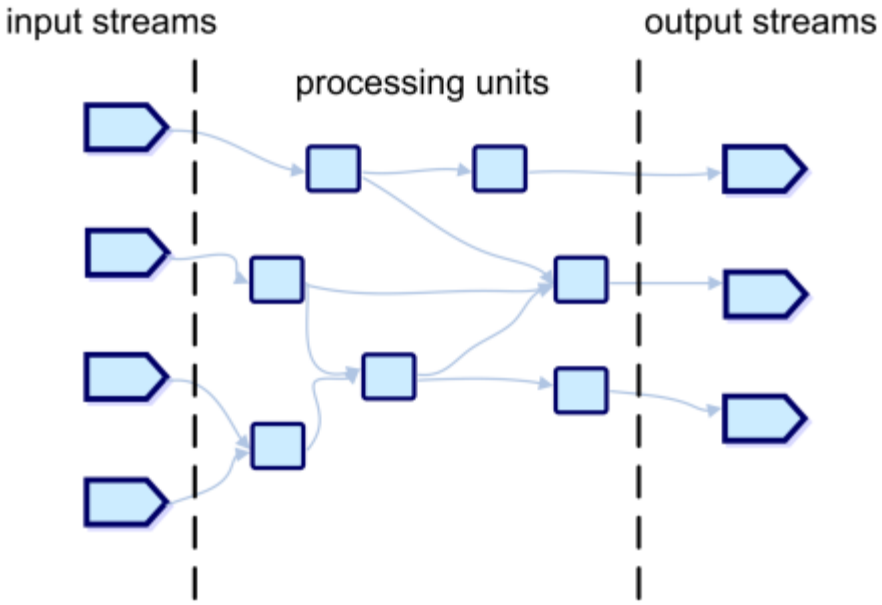
数据：自动到达



控制流计算 V.S. 数据流计算

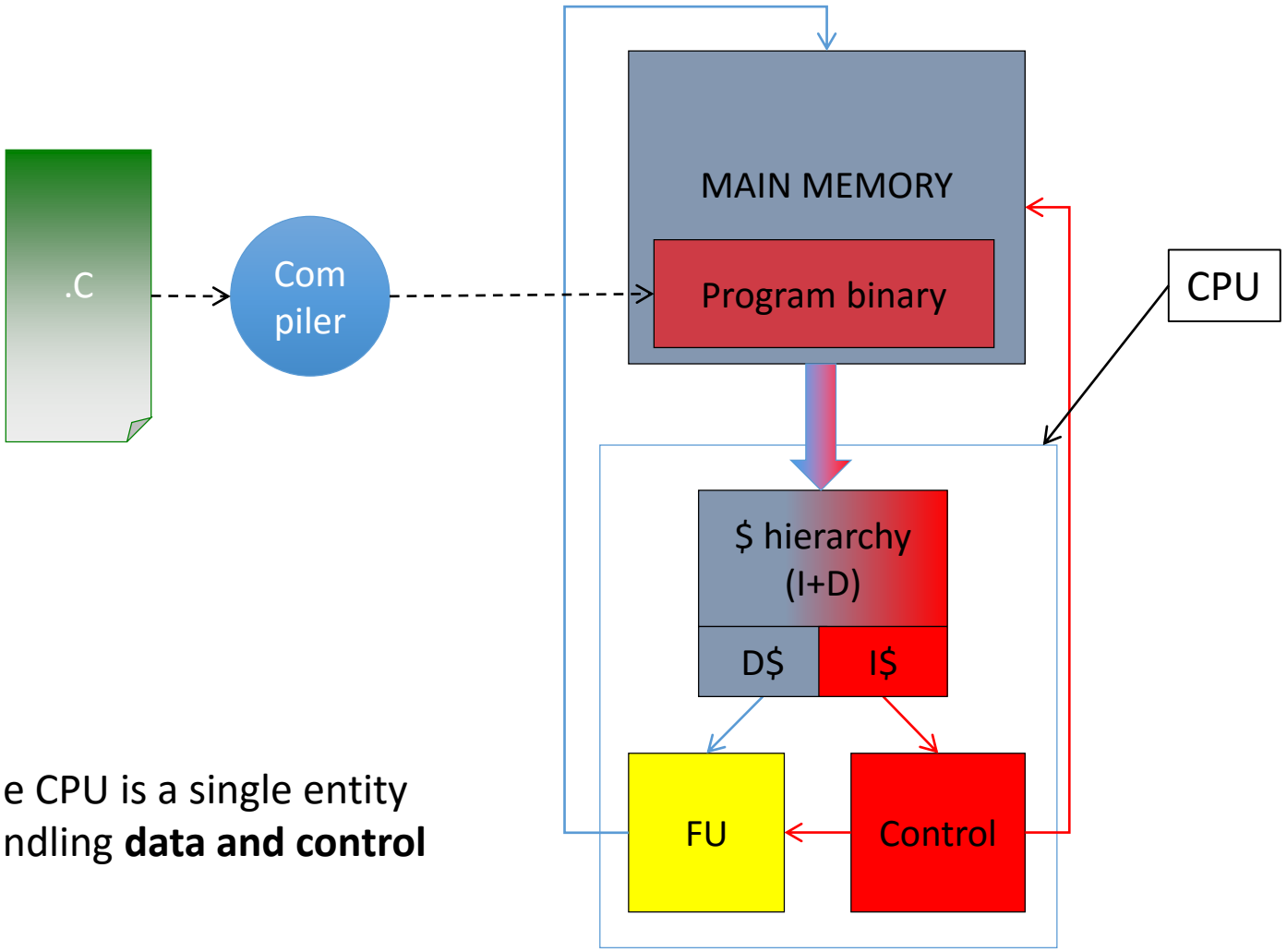
指令：读取、解析、执行　　指令：固定功能





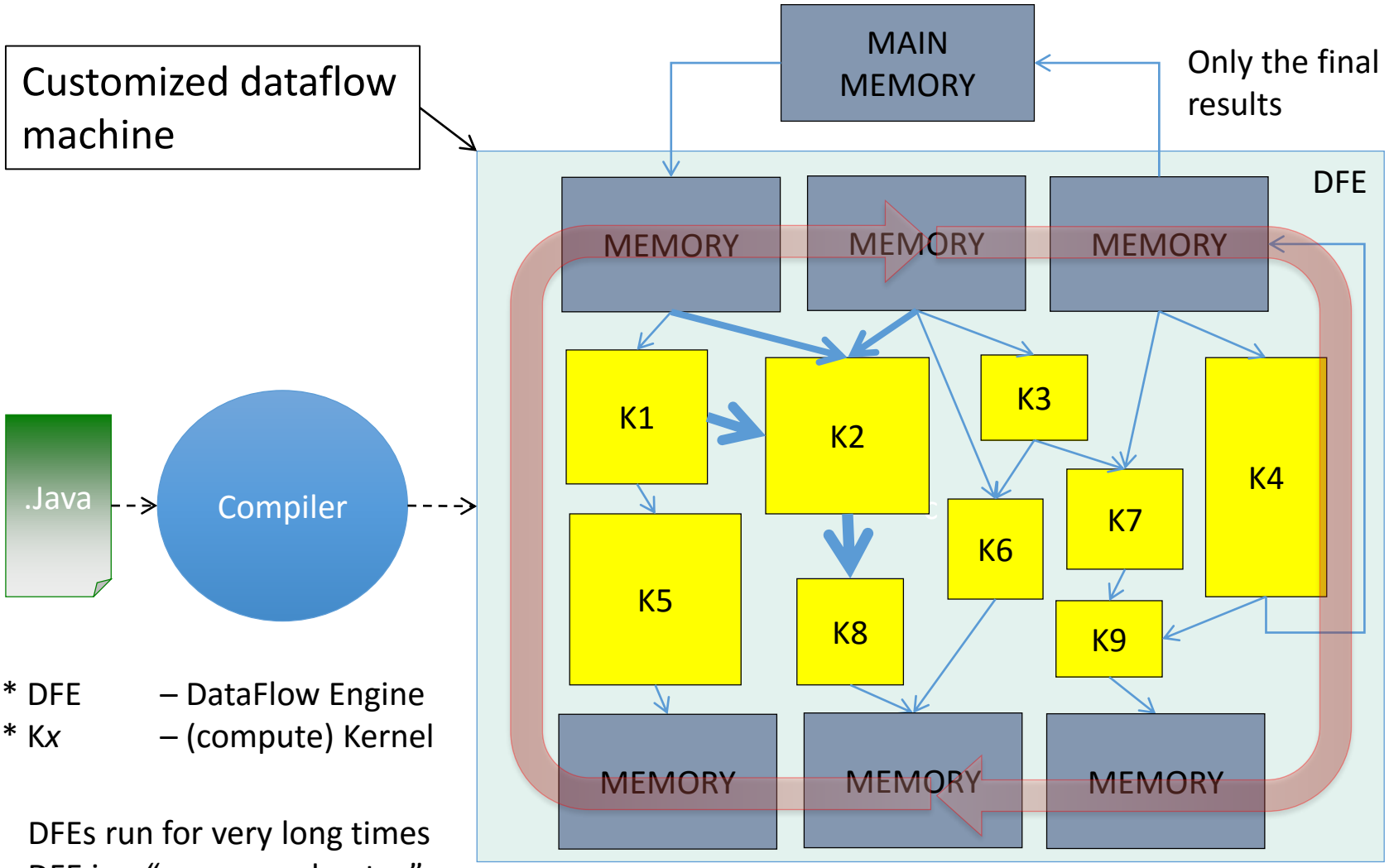
A Practical
Data Flow Design
with 4866 nodes

Control-flow Machine



The CPU is a single entity handling **data and control**

Data Flow Computing Machine



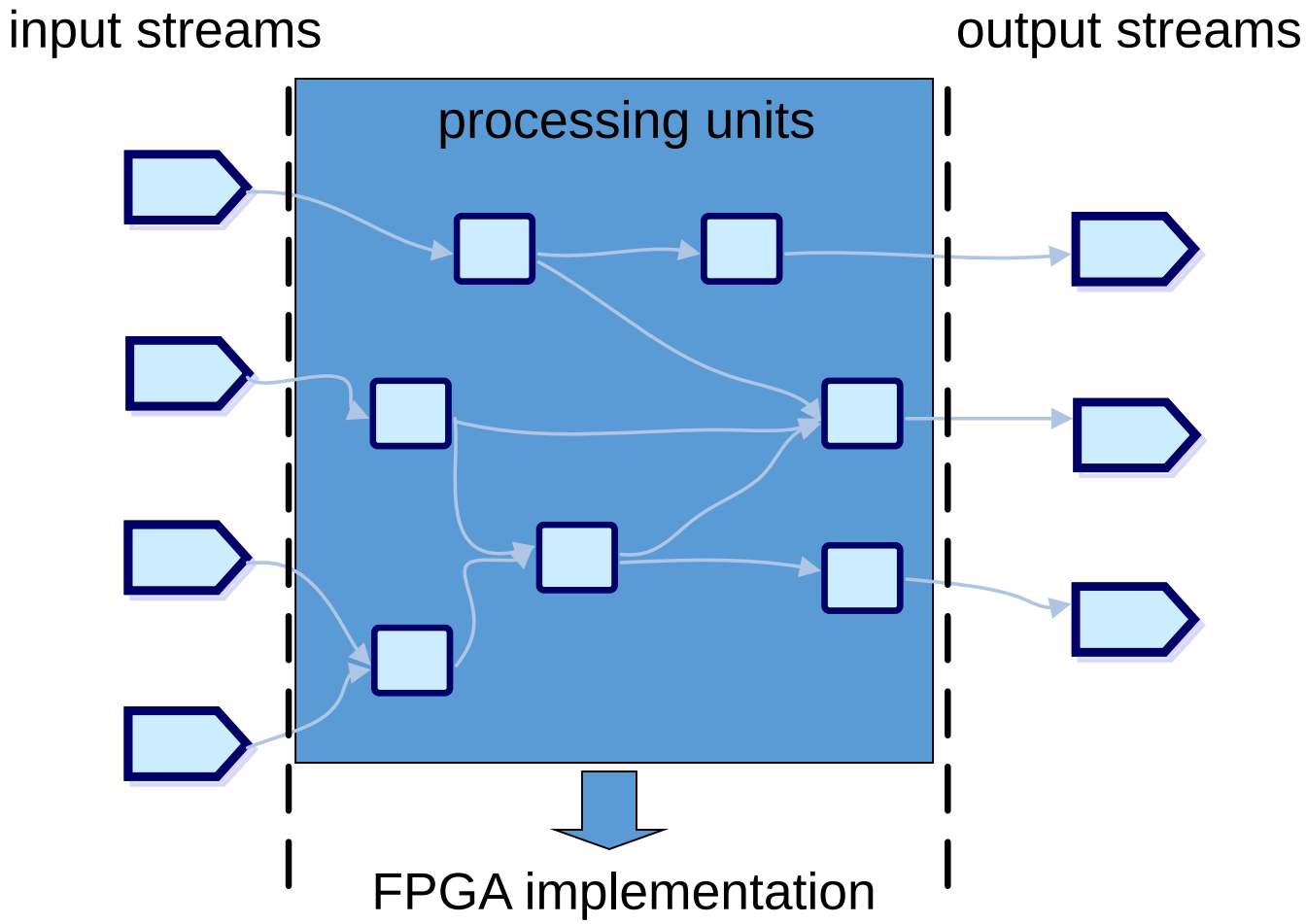
* DFE – DataFlow Engine
* K*x* – (compute) Kernel

DFEs run for very long times
DFE is a “mega accelerator”

本讲提纲

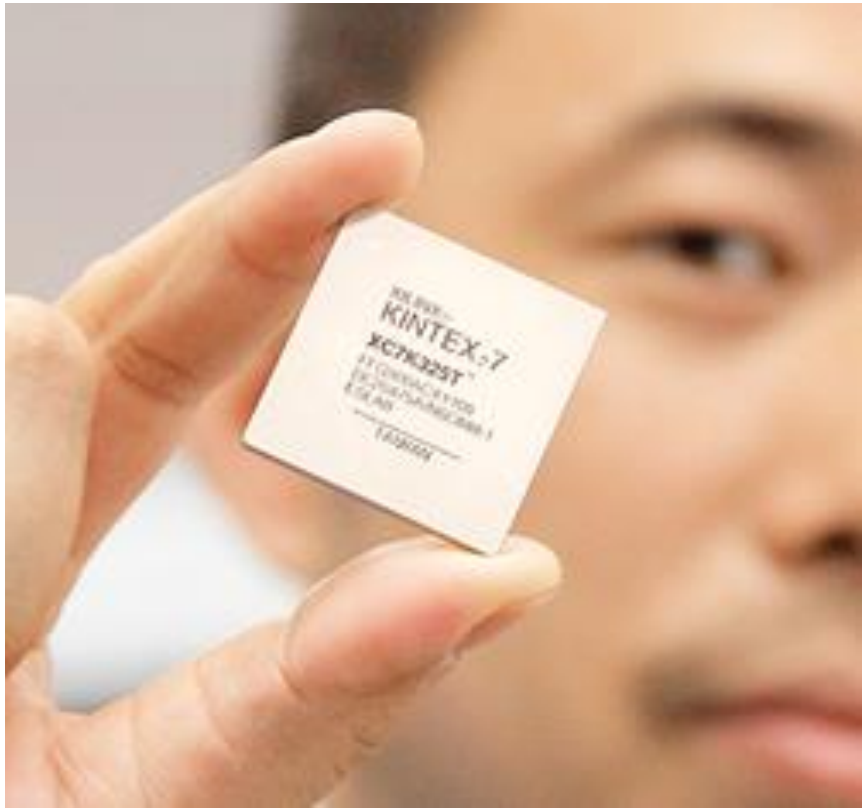
- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限

Data Flow Computing: 如何实现？



什么是FPGA?

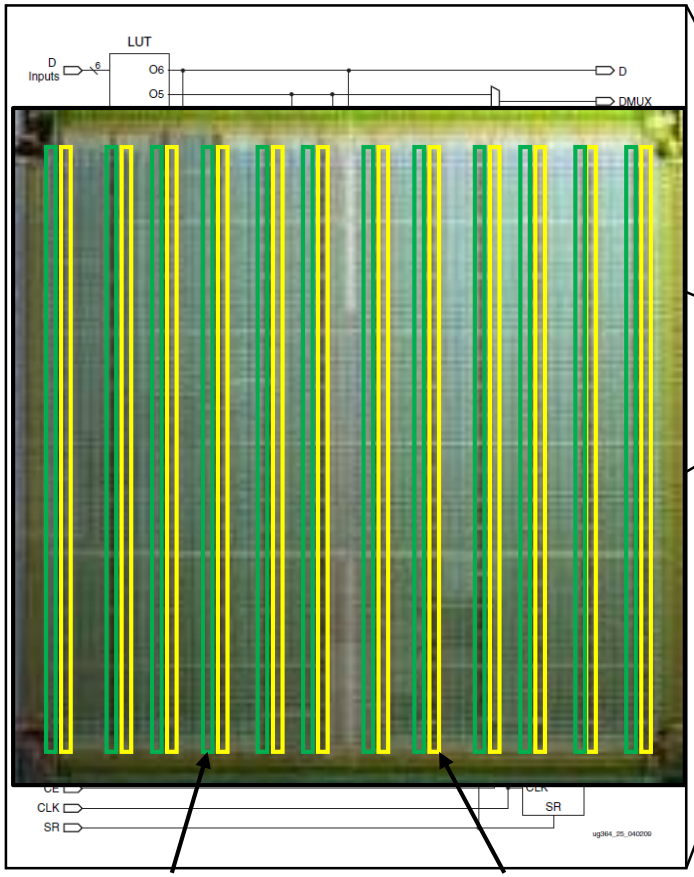
- Field Programmable Gate Arrays
- 现场可编程门阵列



什么是FPGA?

- 可编程的硬件芯片
- 内部的计算器件以及器件之间的连接方式均为用户可控
- 用户可以此为平台进行硬件设计与编程

Logic Cell (10^5 elements)

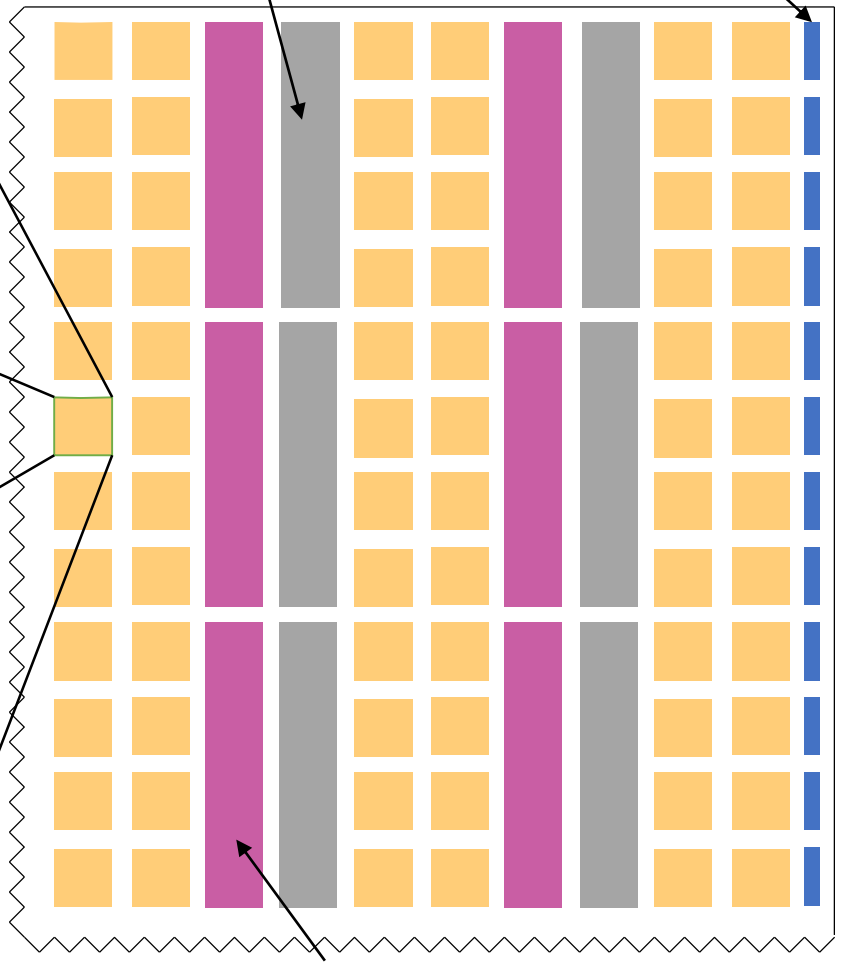


Block RAM

DSP Block

DSP Block

IO Block

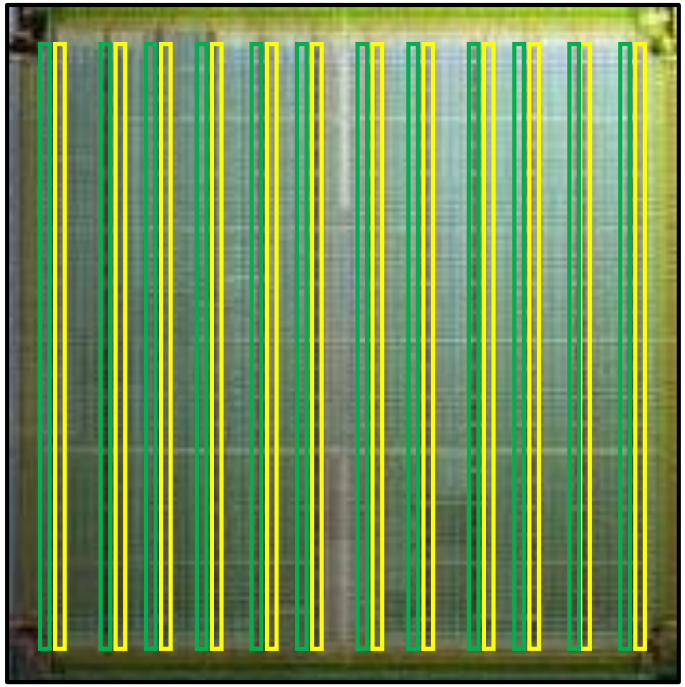


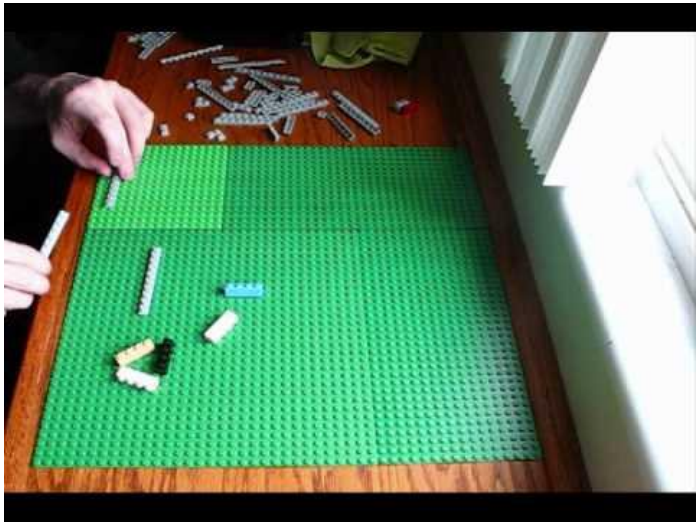
Block RAM (20TB/s)

Xilinx Virtex-6 FPGA

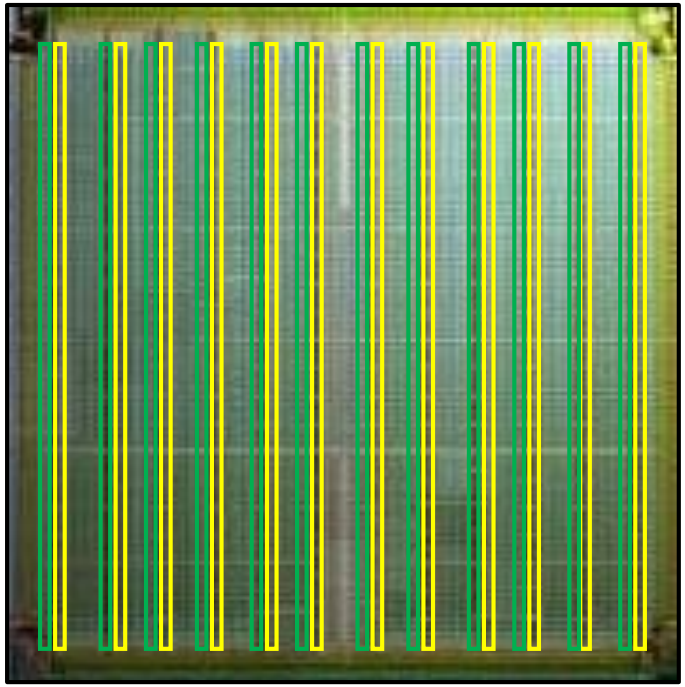
FPGA V.S. LEGO

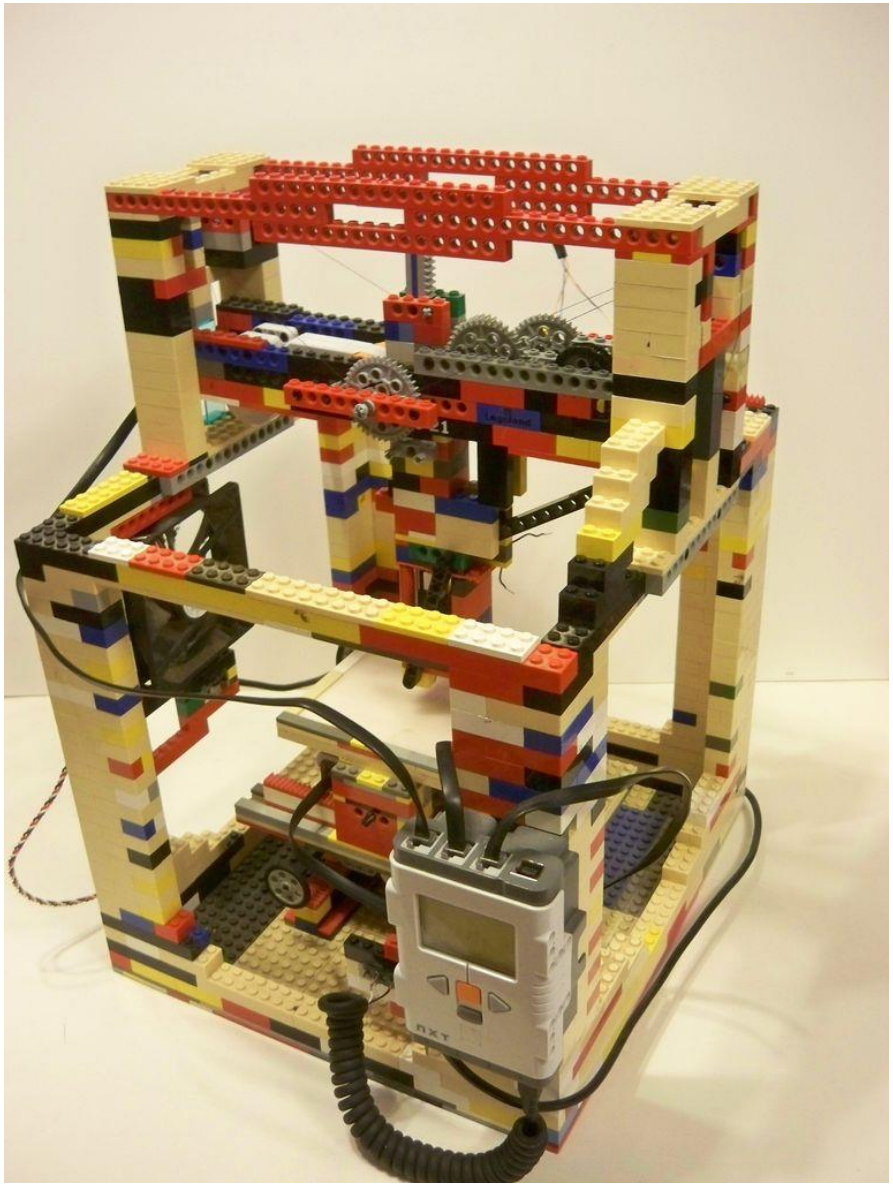
FPGA V.S. LEGO



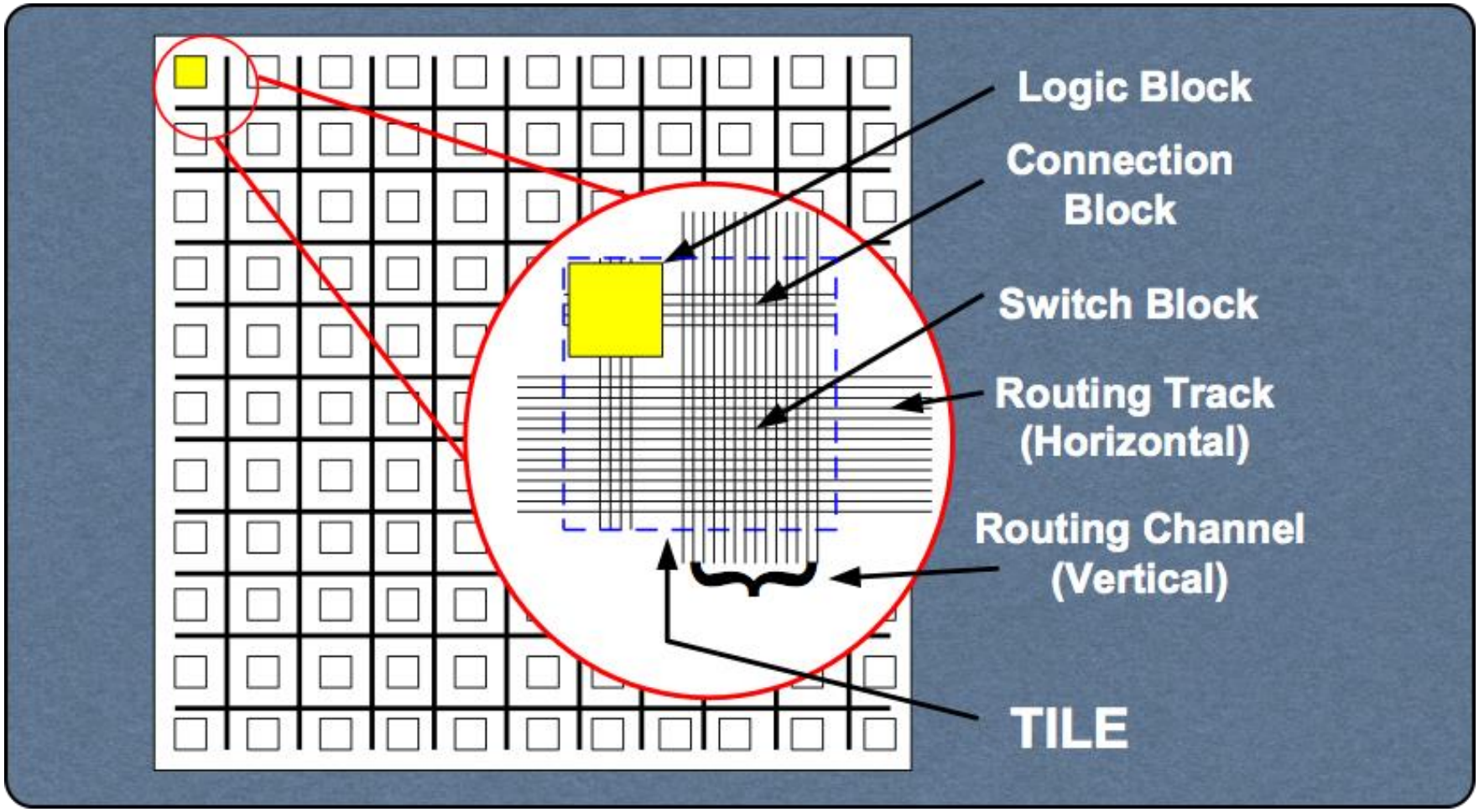


FPGA V.S. LEGO



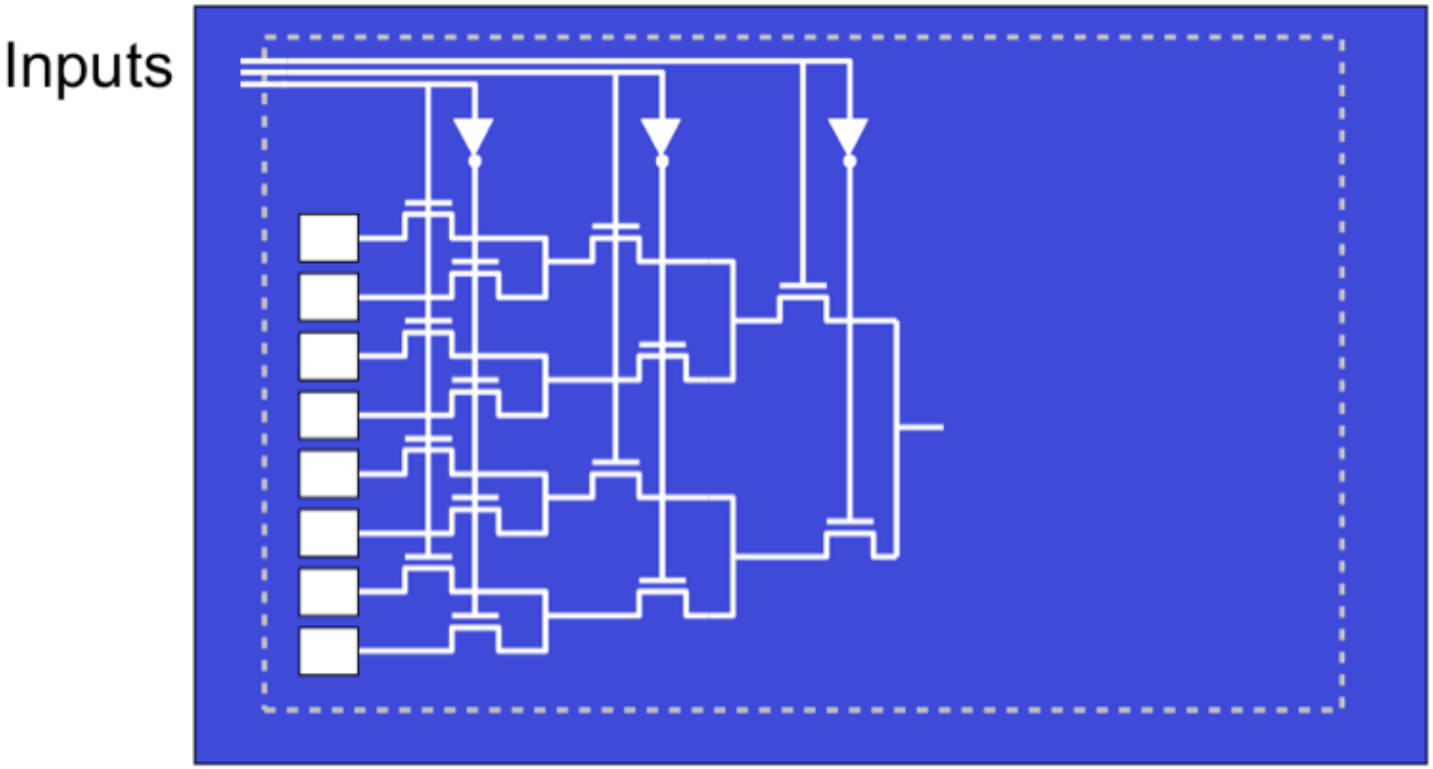


FPGA芯片内部结构


Logic Block
Connection Block
Switch Block
Routing Track (Horizontal)
Routing Channel (Vertical)
TILE

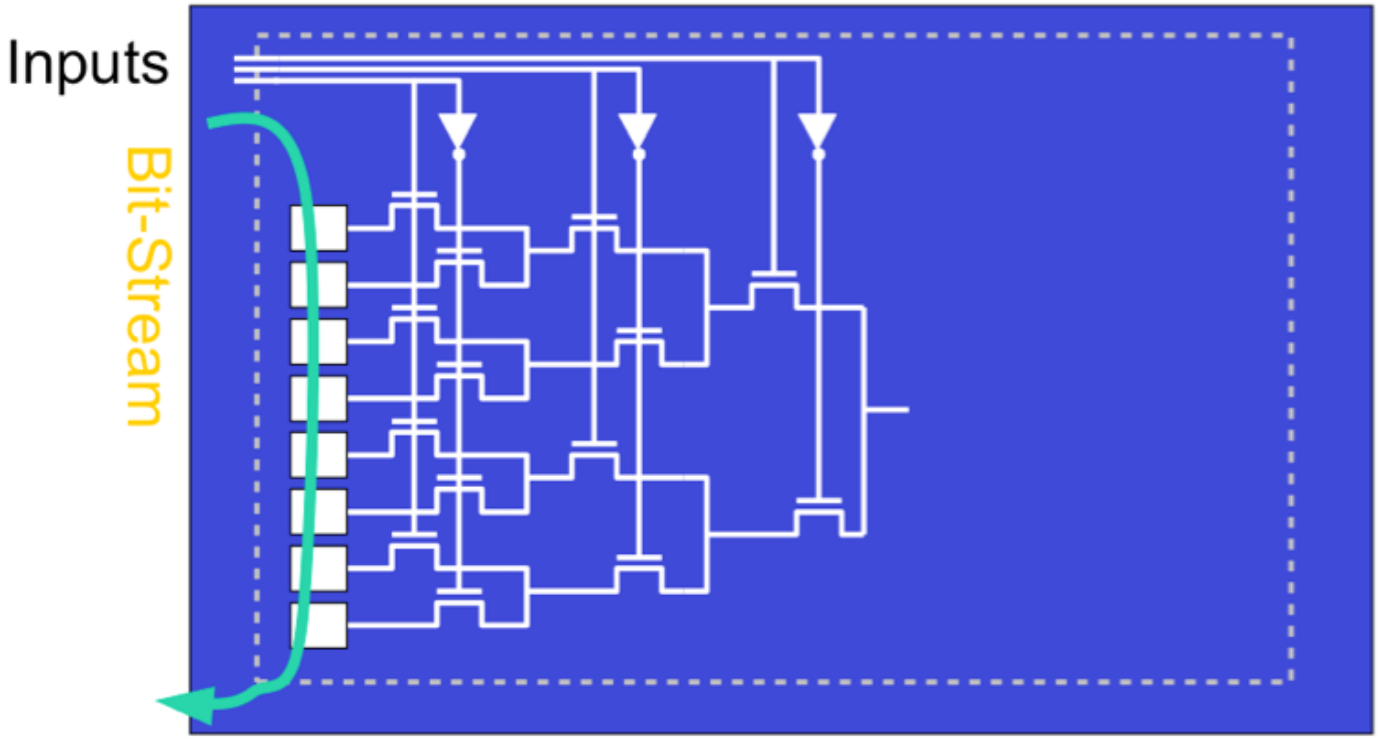
FPGA芯片内部结构

- 可编程的最基本模块：lookup table



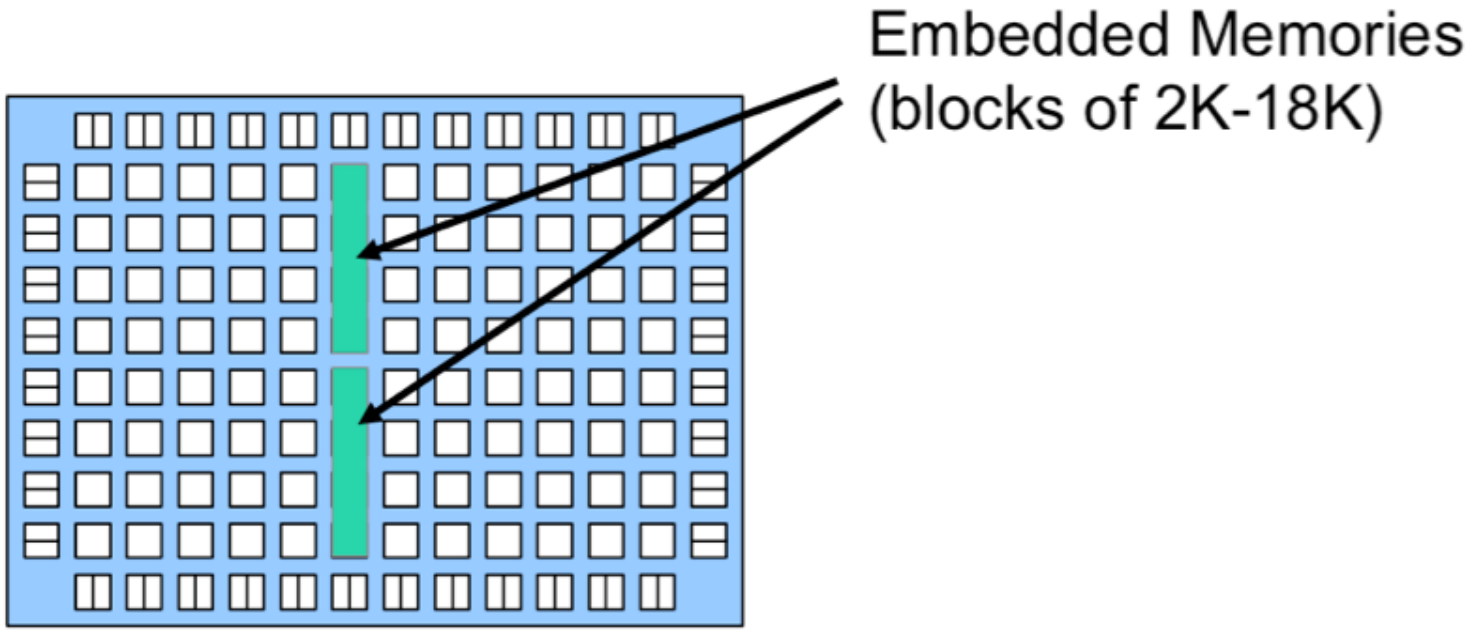
FPGA芯片内部结构

- 可编程的最基本模块：lookup table



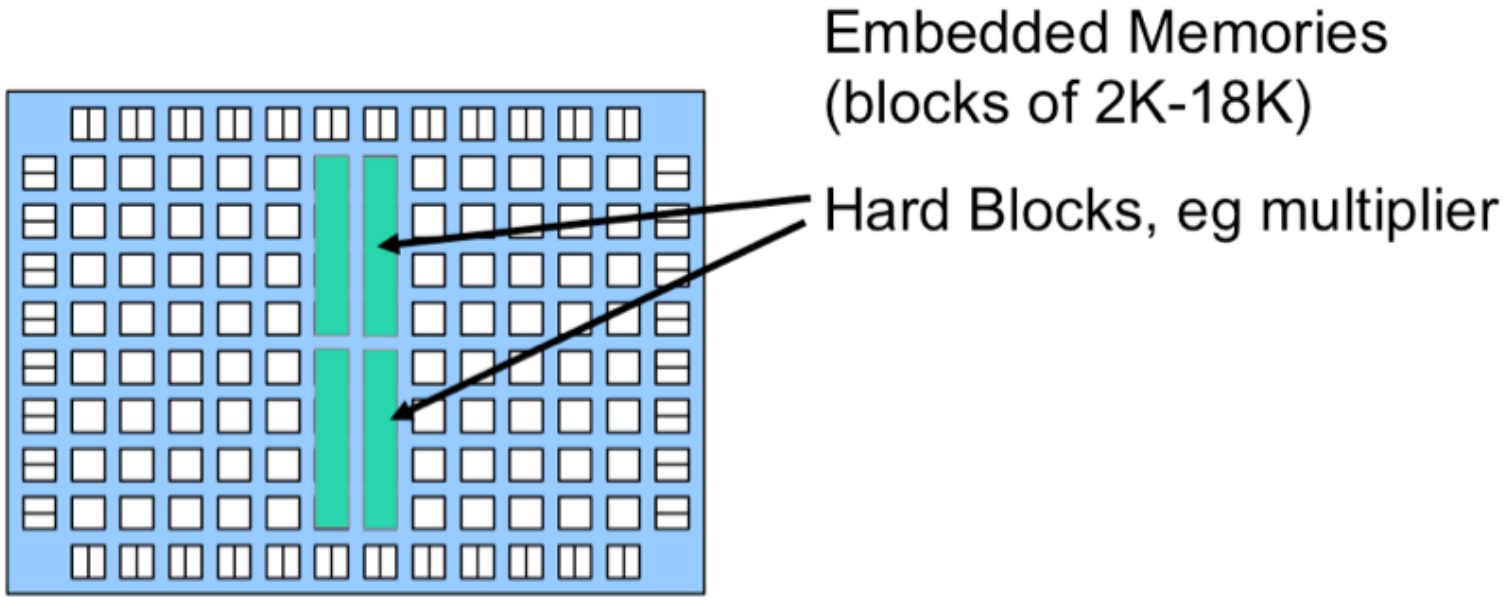
FPGA芯片内部结构

- 引入块状存储



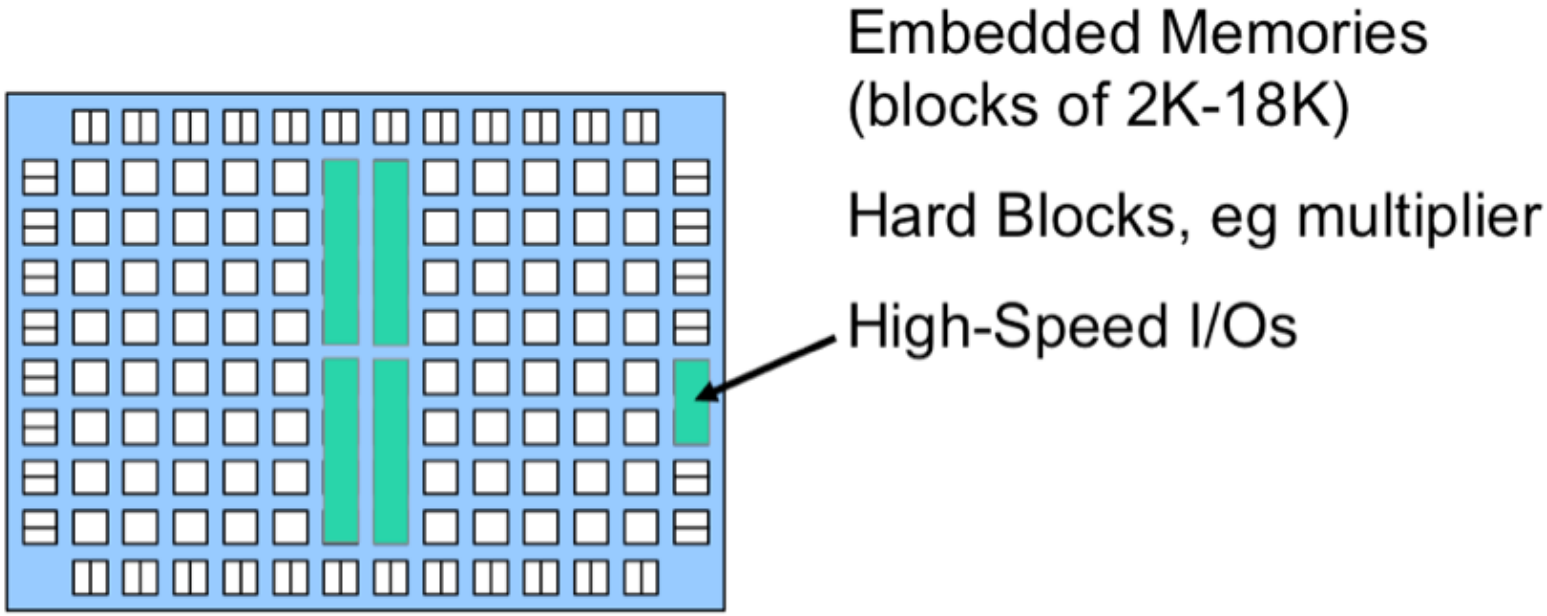
FPGA芯片内部结构

- 引入乘法器



FPGA芯片内部结构

- 引入高速I/O



FPGA的计算能力

- 以Xilinx Virtex 7为例：
- 1,955 K logic cells
- 3,600 DSP
- 1500+ MULTs, 1500+ ADDs

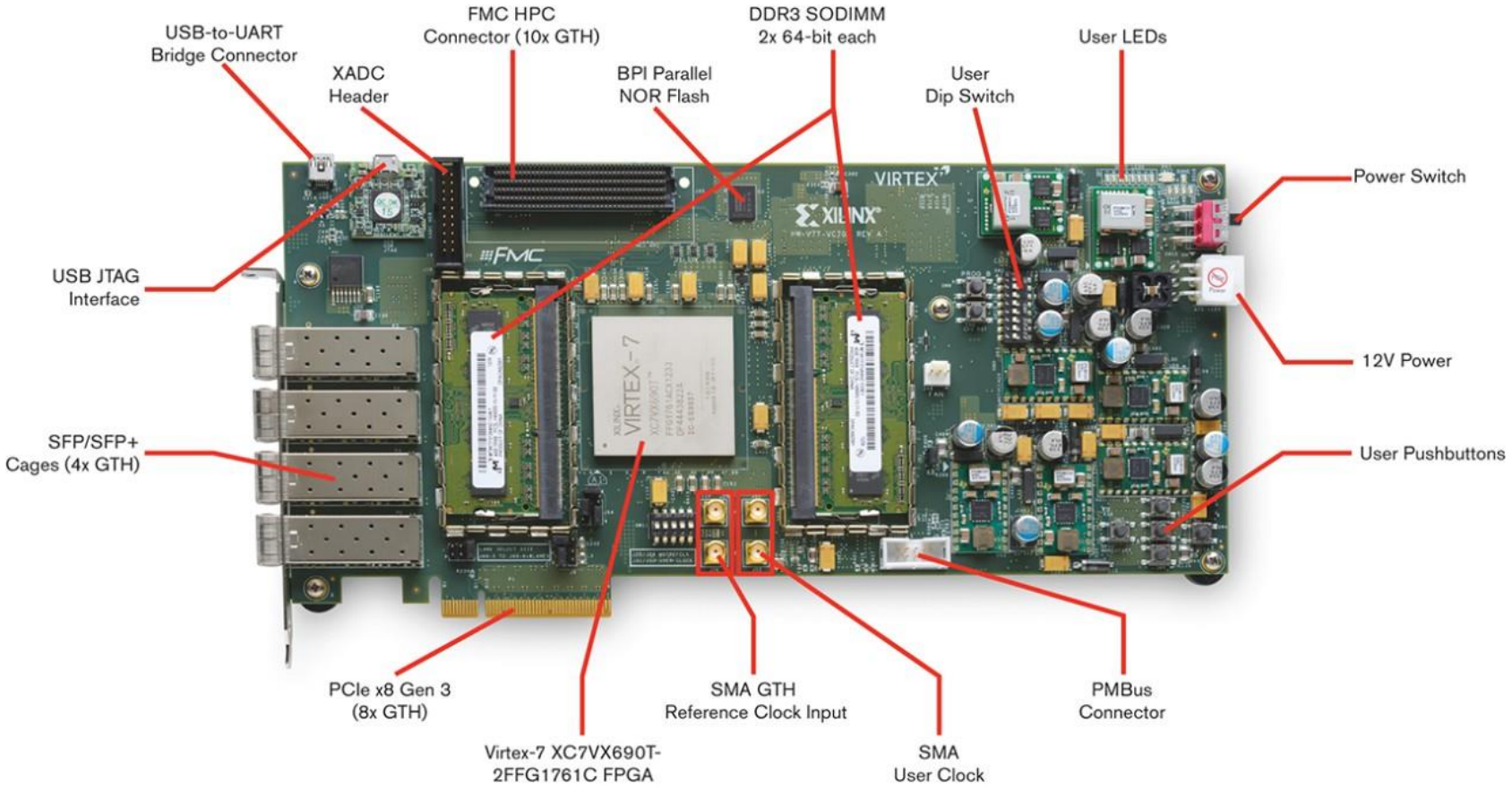
FPGA的缓存机制

- 以Xilinx Virtex 7为例：
- 68 Mb fast storage
- 3,760 18-Kb block RAMs
- 3,760 x 32 bit/s x 2 x 150 MHz
  = 36 Tbit/s = 4.5 TB/s

现有FPGA平台举例：Xilinx Virtex-7 VC709开发板

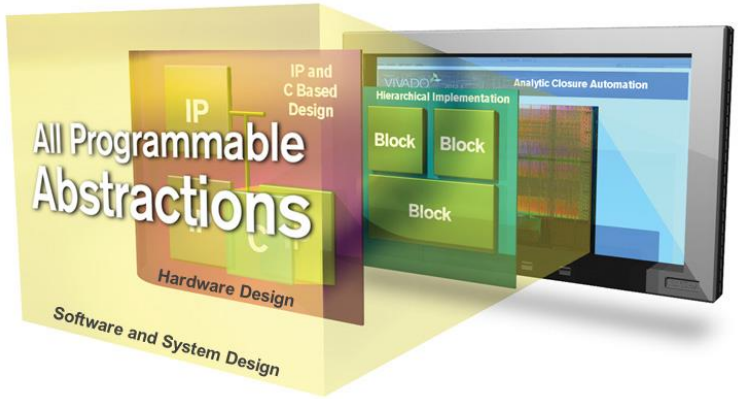
- Xilinx Vitex-7 VX690T FPGA
- 4x 万兆网接口
- FMC接口：多媒体处理

现有FPGA平台举例：Xilinx Virtex-7 VC709开发板



现有FPGA平台举例：Xilinx Virtex-7 VC709开发板

- Vivado 设计套件：



- 加速实现
  - 设计实现时间缩短 4 倍
  - 设计密度提升 20%
  - 高达 3 速度级的性能优势; 功耗降低 35%

- 加速集成
  - 基于C 的 IP 和 VIVADO HLS 集成
  - 基于模型的 DSP 设计和 System Generator for DSP 集成
  - 基于模块的 IP 和 VIVADO IP Integrator 集成

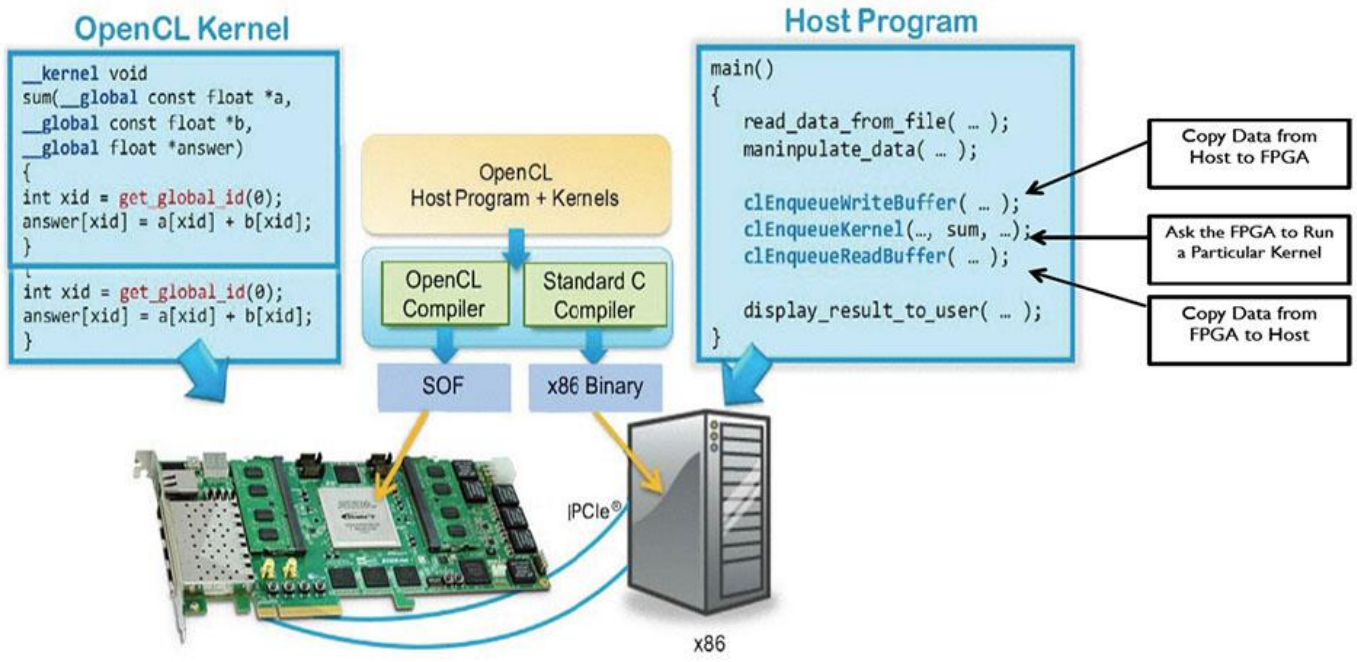
- 加速验证
  - 面向设计和仿真的集成设计环境
  - 全面的硬件调试
  - 使用 C、C++ 或 SystemC 加速验证超过 100 倍

现有FPGA平台举例： Nallatech PCIe-395

- Altera Stratix V AB FPGA
- 4x 32G 内存
- 4x 万兆网接口

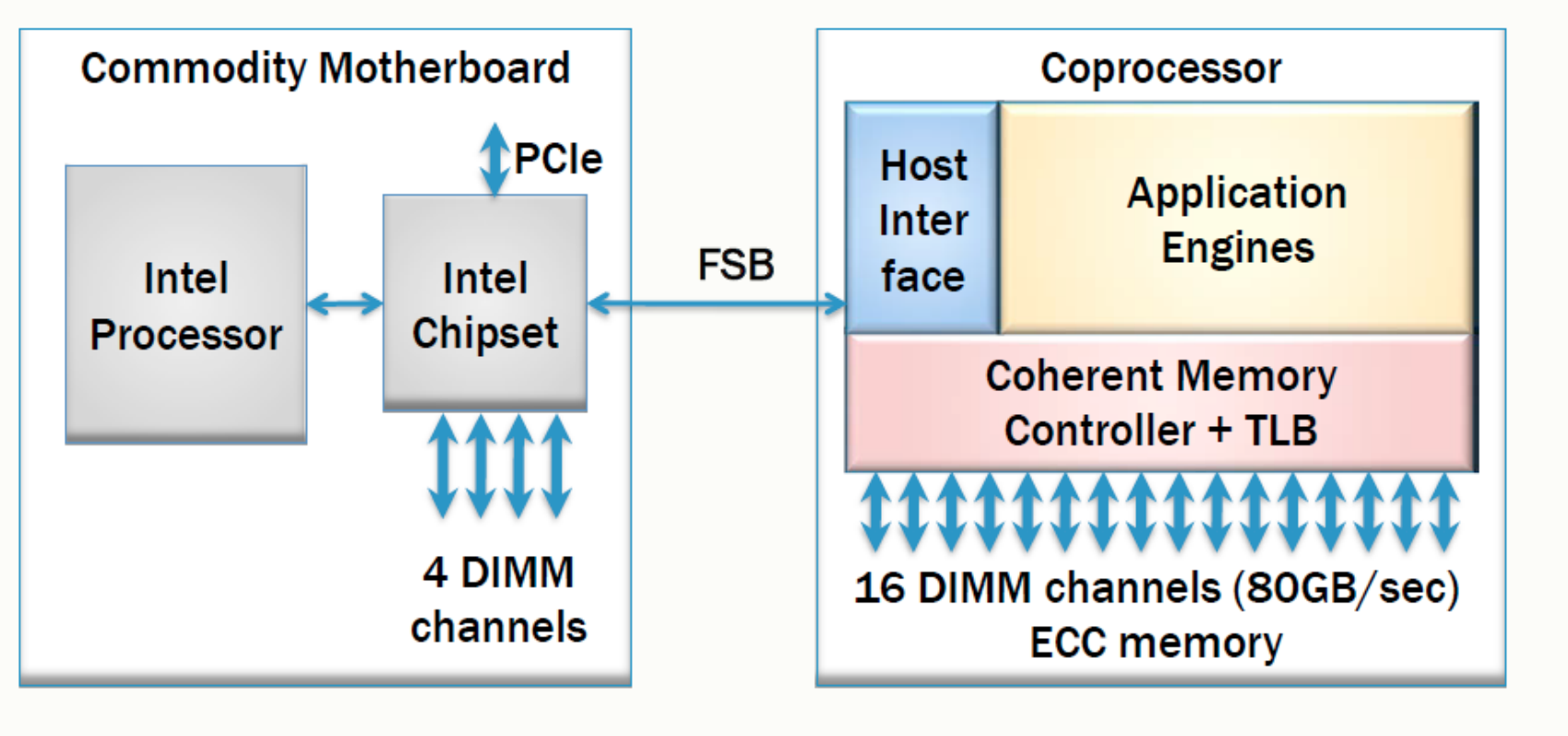
现有FPGA平台举例：Nallatech PCIe-395

- OPENCL编程套件



现有FPGA平台举例：Convey Computer

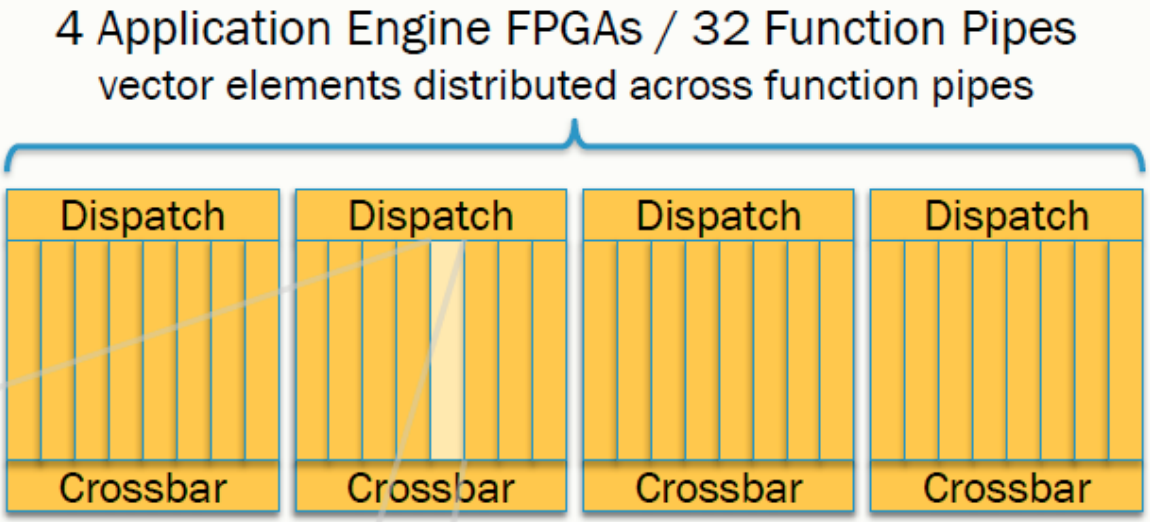
- FPGA作为CPU的coprocessor

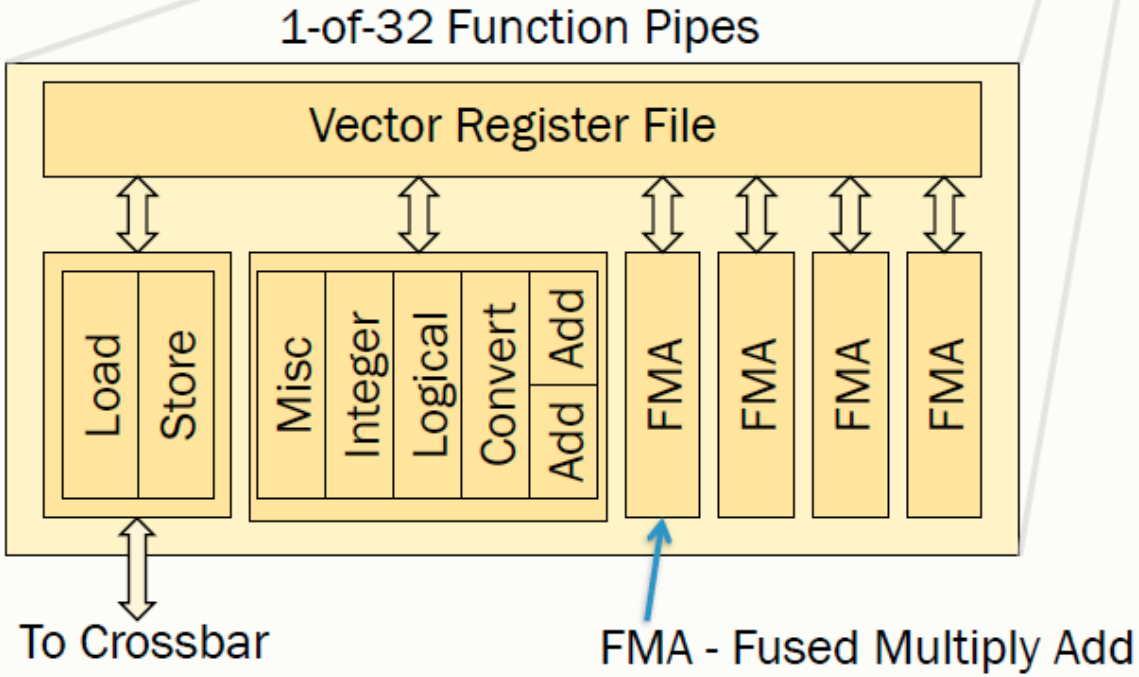


Single Precision Vector Personality

Load-store vector architecture with modern latency-hiding features

Optimized for Signal Processing applications





A function pipe is the unit of data path replication

Same instructions sent to all function pipes (SIMD)

Each function pipe supports:

- Multiple functional units
- Out-of-order execution
- Register renaming

CONVEY computer™

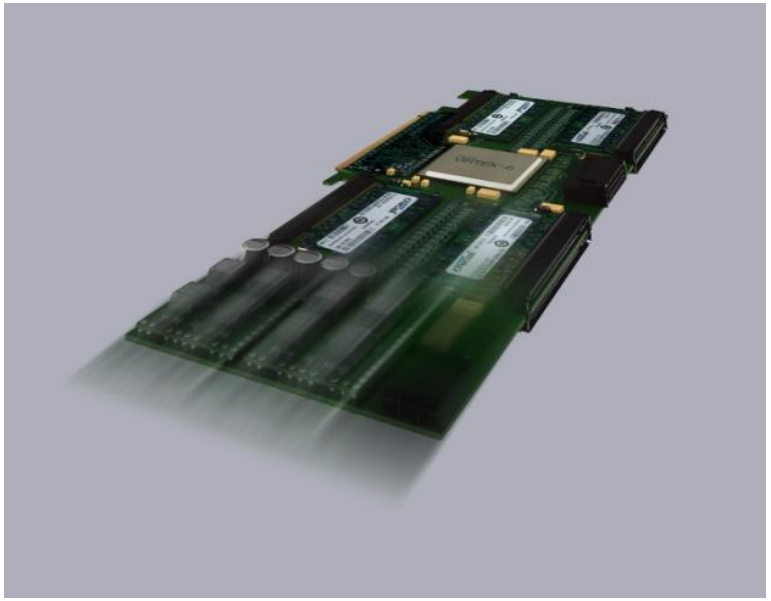
```
!$cny begin_coproc
    call stencil_sub(p1,p2,NX,NY,NZ,sp,sx1,sx2,sy1,sy2,sz1,sz2)
!$cny end_coproc
```

- 编程方式：

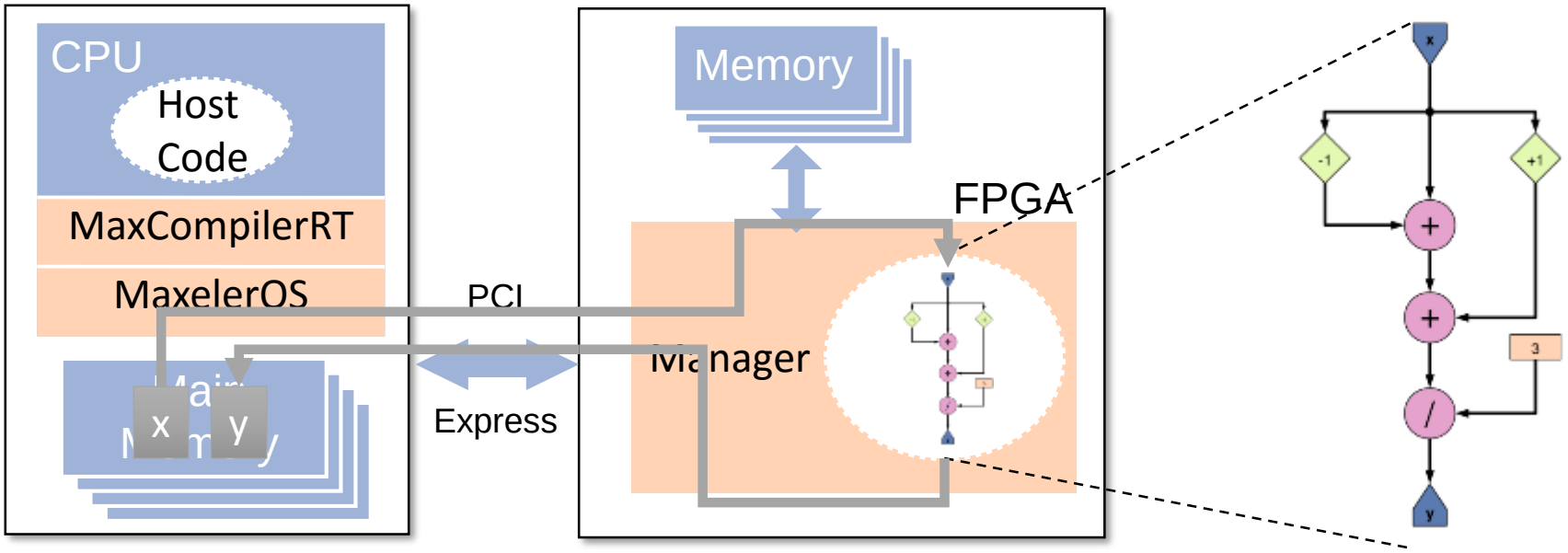
```
subroutine stencil_sub(p1,p2,NX,NY,NZ,sp,sx1,sx2,sy1,sy2,sz1,sz2)
  implicit none
  integer*8 NX,NY,NZ,i,j,k
  real*4 p1(NX,NY,NZ),p2(NX,NY,NZ), sp,sx1,sx2,sy1,sy2,sz1,sz2
  do k=2,NZ-1
    do j=2,NY-1
      do i=2,NX-1
        p2(i,j,k) = sp*p1(i,j,k) + sx1*p1(i-1,j  ,k  )      &
                                 + sx2*p1(i+1,j  ,k  )      &
                                 + sy1*p1(i  ,j-1,k  )      &
                                 + sy2*p1(i  ,j+1,k  )      &
                                 + sz1*p1(i  ,j  ,k-1)      &
                                 + sz2*p1(i  ,j  ,k+1)

      enddo
    enddo
  enddo
  return
```

现有FPGA平台举例：Maxeler DFE (data flow engine)

- Maxeler Max3 Acceleration Card
  - one Xilinx Virtex 6 SX475T
    - 74,400 logic slices
    - 1064 36Kb Block RAMs
    - 2016 25x18 DSPs
  - 48 GB onboard memory
  - PCIE connection to host
  - MaxRing connection between cards



现有FPGA平台举例：Maxeler DFE: MaxCompiler编程套件



Host Code (.c)

```
float *x, *y;
for (int i =1; i < DATA_SIZE-1; i++)
   y[i]=(x[i-1]+x[i]+x[i+1])/3;
max_run(device,
   max_input("x", x, DATA_SIZE*4),
   max_output("y", y, DATA_SIZE*4),
   max_runfor("Kernel", DATA_SIZE));
```

Manager (.java)

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

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

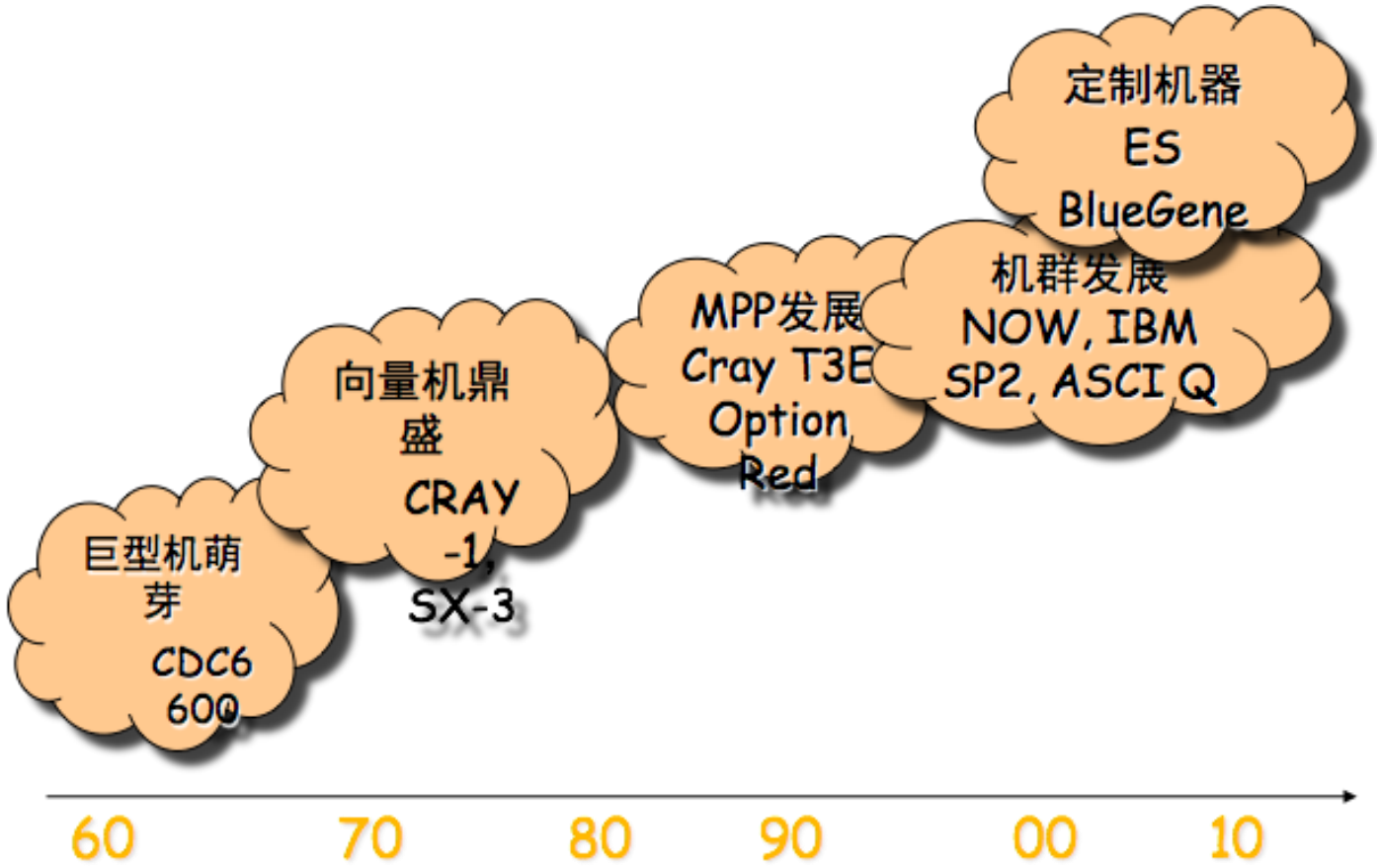
MovingAverageKernel (.java)

```
HWVar x = io.input("x", hwFloat(8, 24));
HWVar prev = stream.offset(x, -1);
HWVar next = stream.offset(x, 1);

HWVar sum = prev+x+next;
HWVar result = sum/3;

io.output("y", result, hwFloat(8, 24));
```

本讲提纲

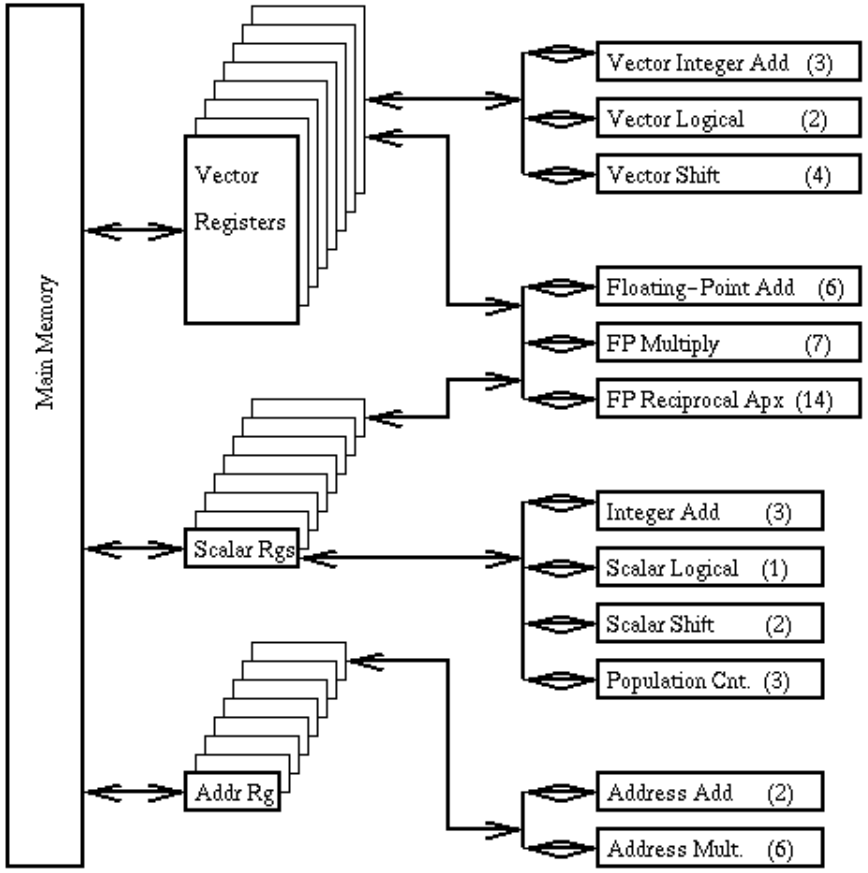
- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限

高性能计算机的发展历程

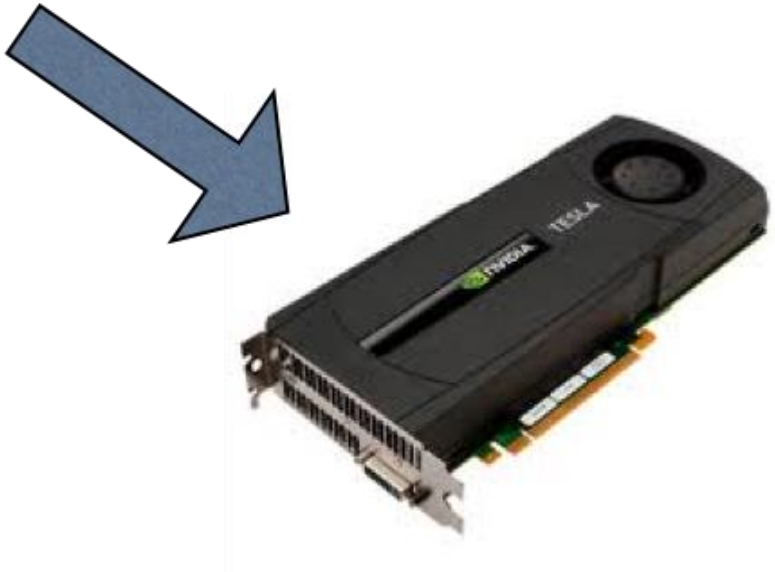


Cray-I 高性能计算机

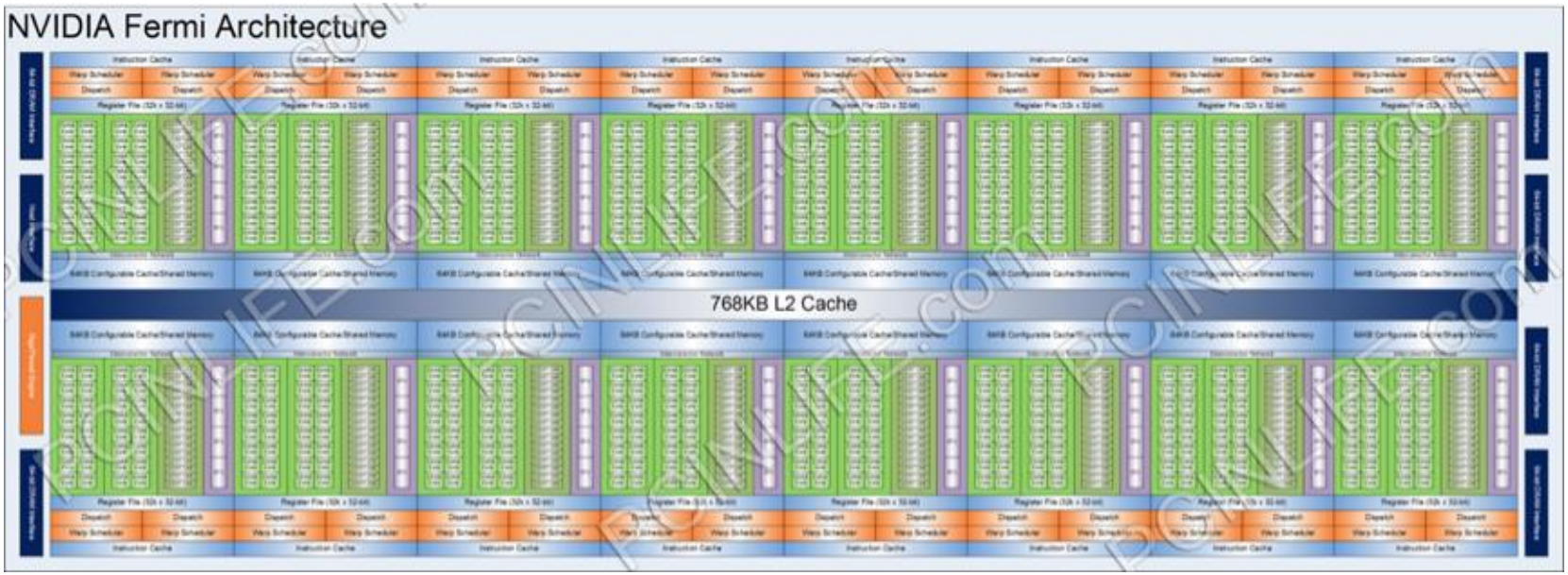
- 向量处理器
- 向量寄存器



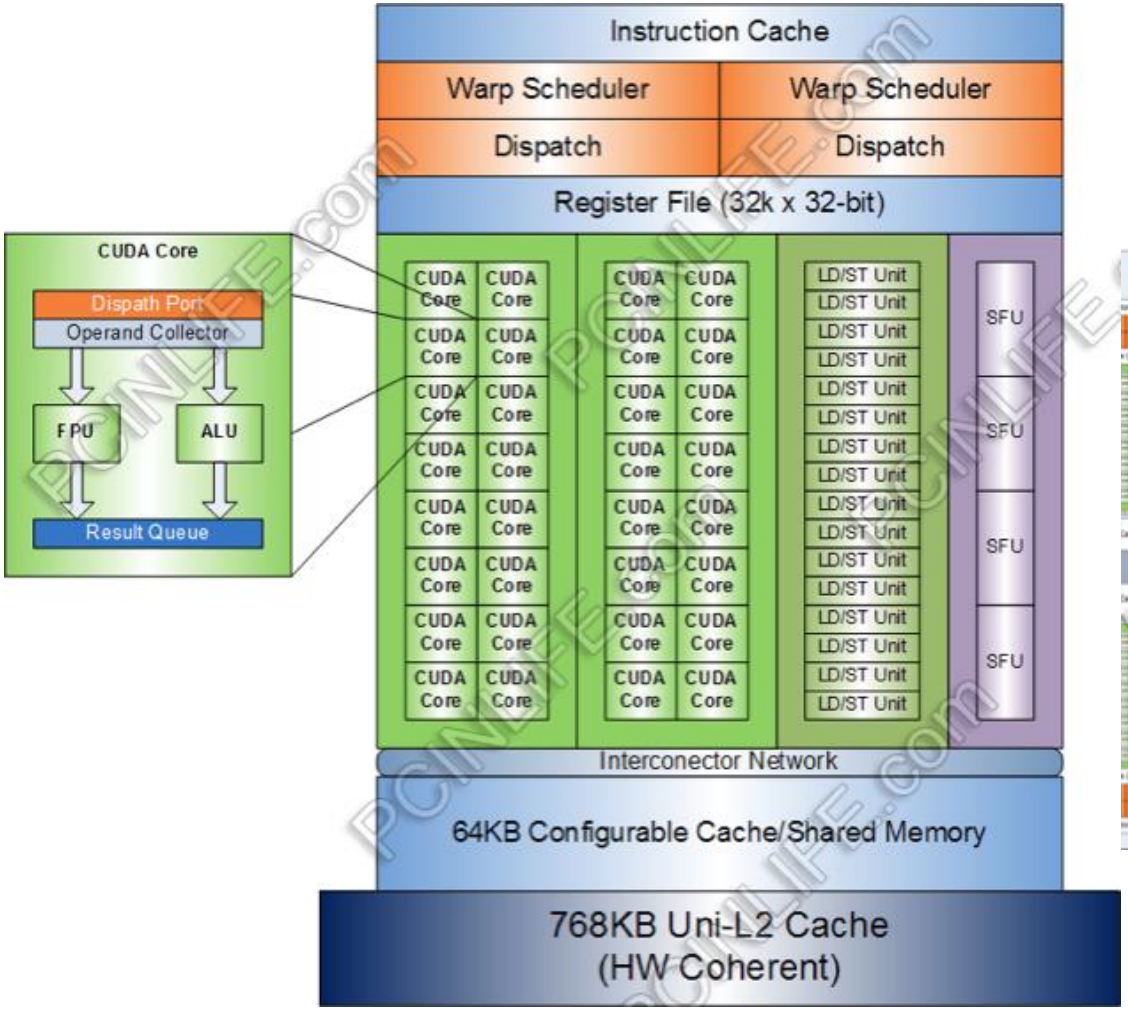
五十年之后

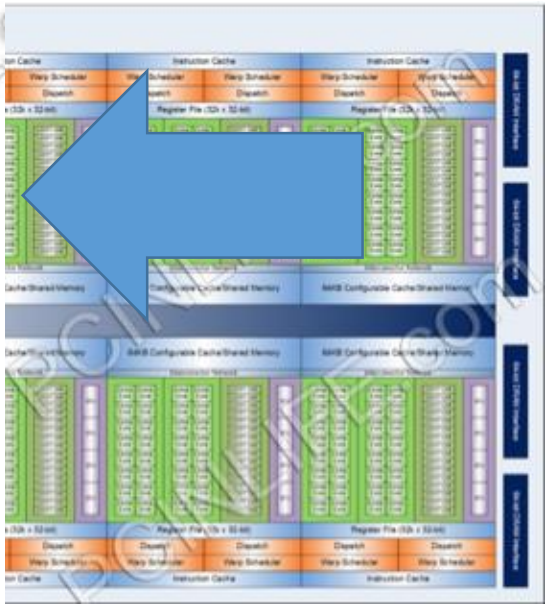


GPU架构



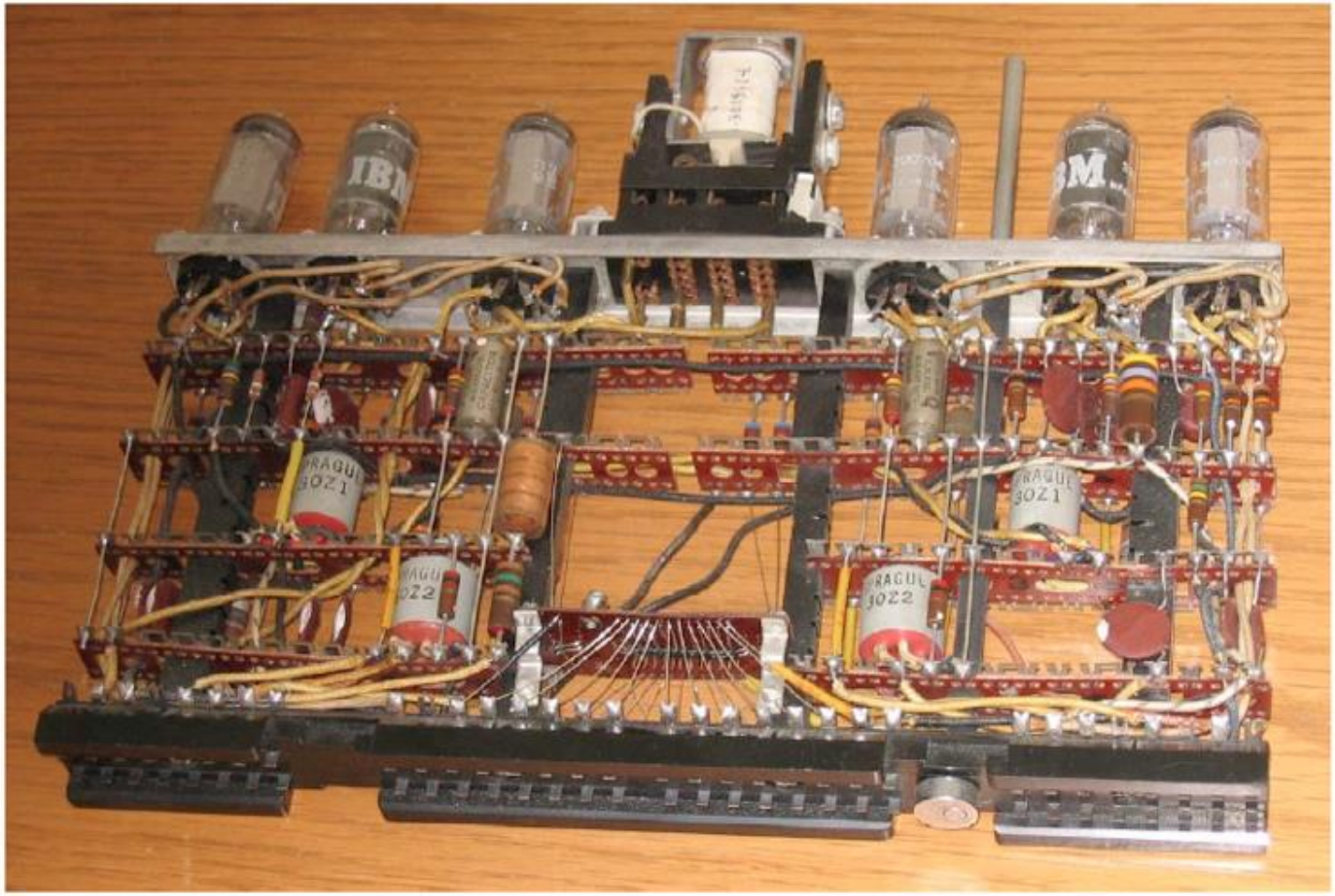
GPU架构


CUDA Core
Dispath Port
Operand Collector
FPU
ALU
Result Queue
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch
Dispatch
Register File (32k x 32-bit)
CUDA Core
LD/ST Unit
SFU
Interconector Network
64KB Configurable Cache/Shared Memory
768KB Uni-L2 Cache
(HW Coherent)



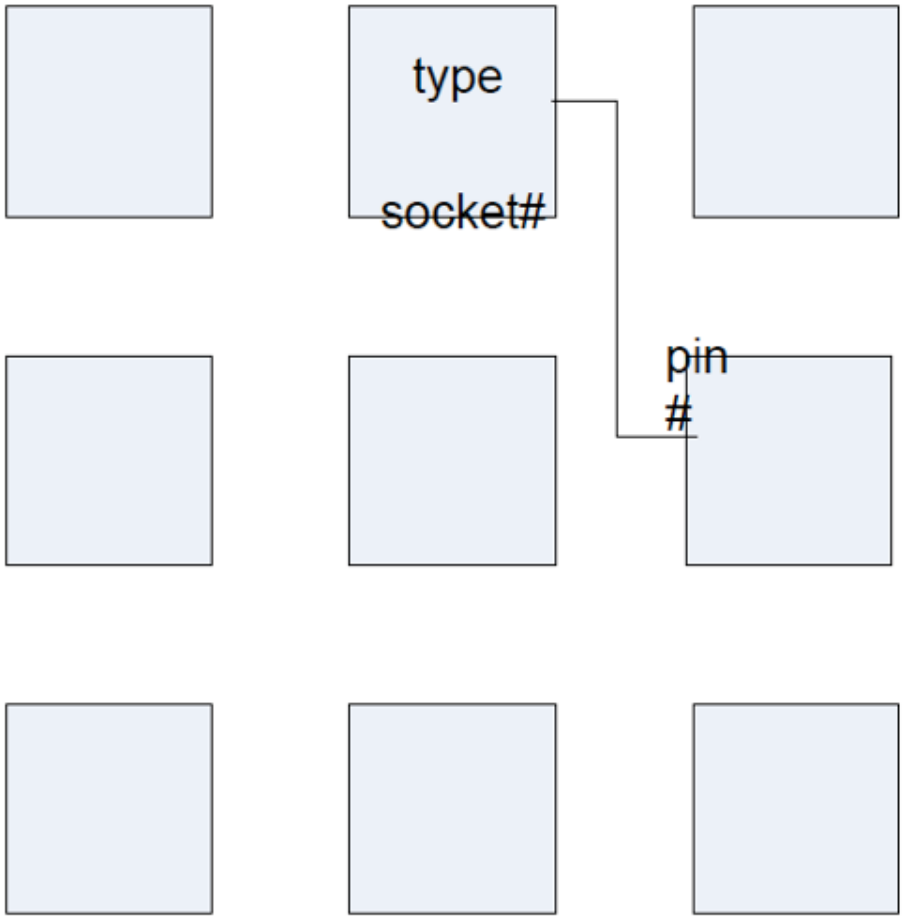
回溯到向量机之前。。。

704 / 709 (c.1955) hand assembly

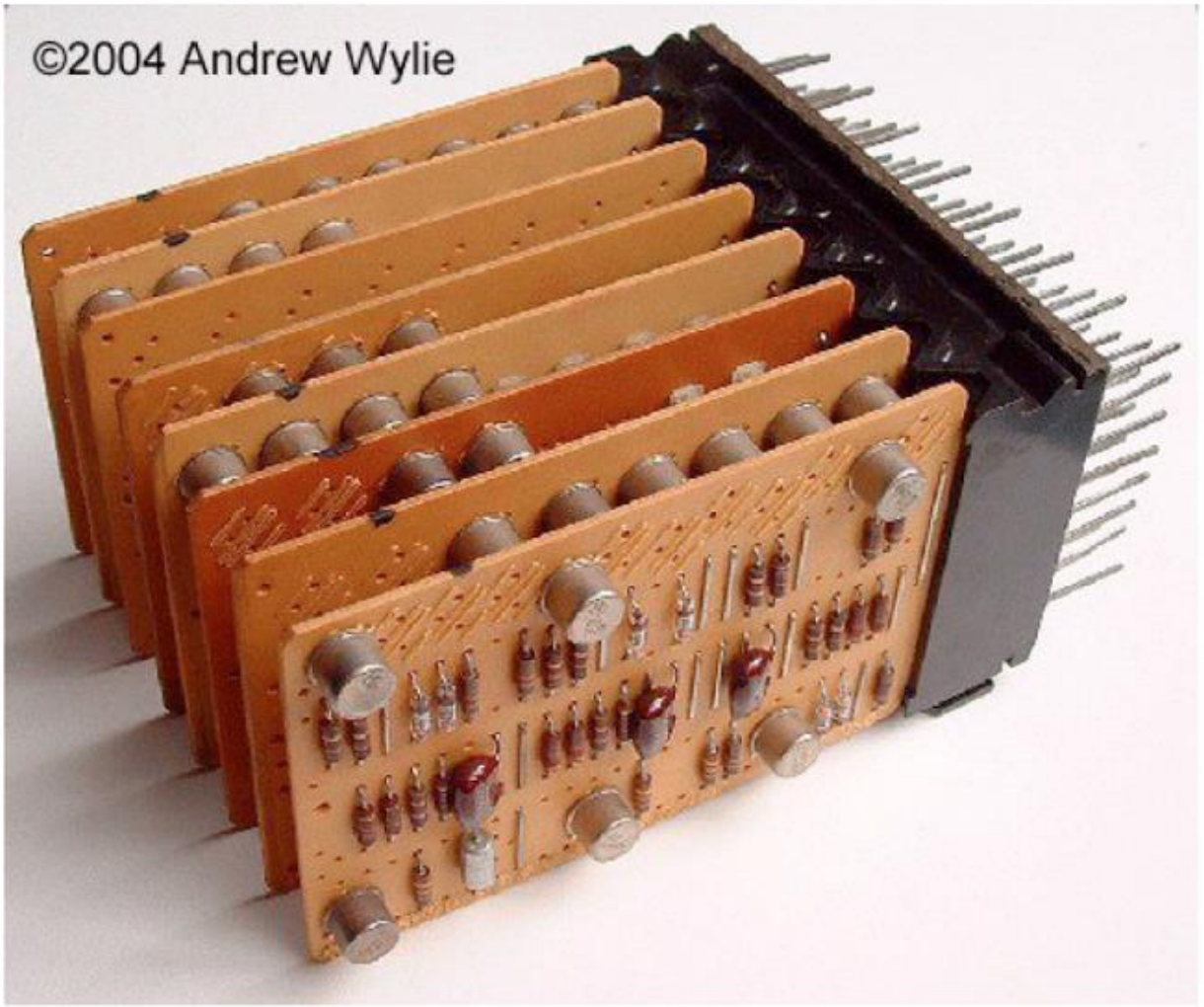


The Standard SMS Card

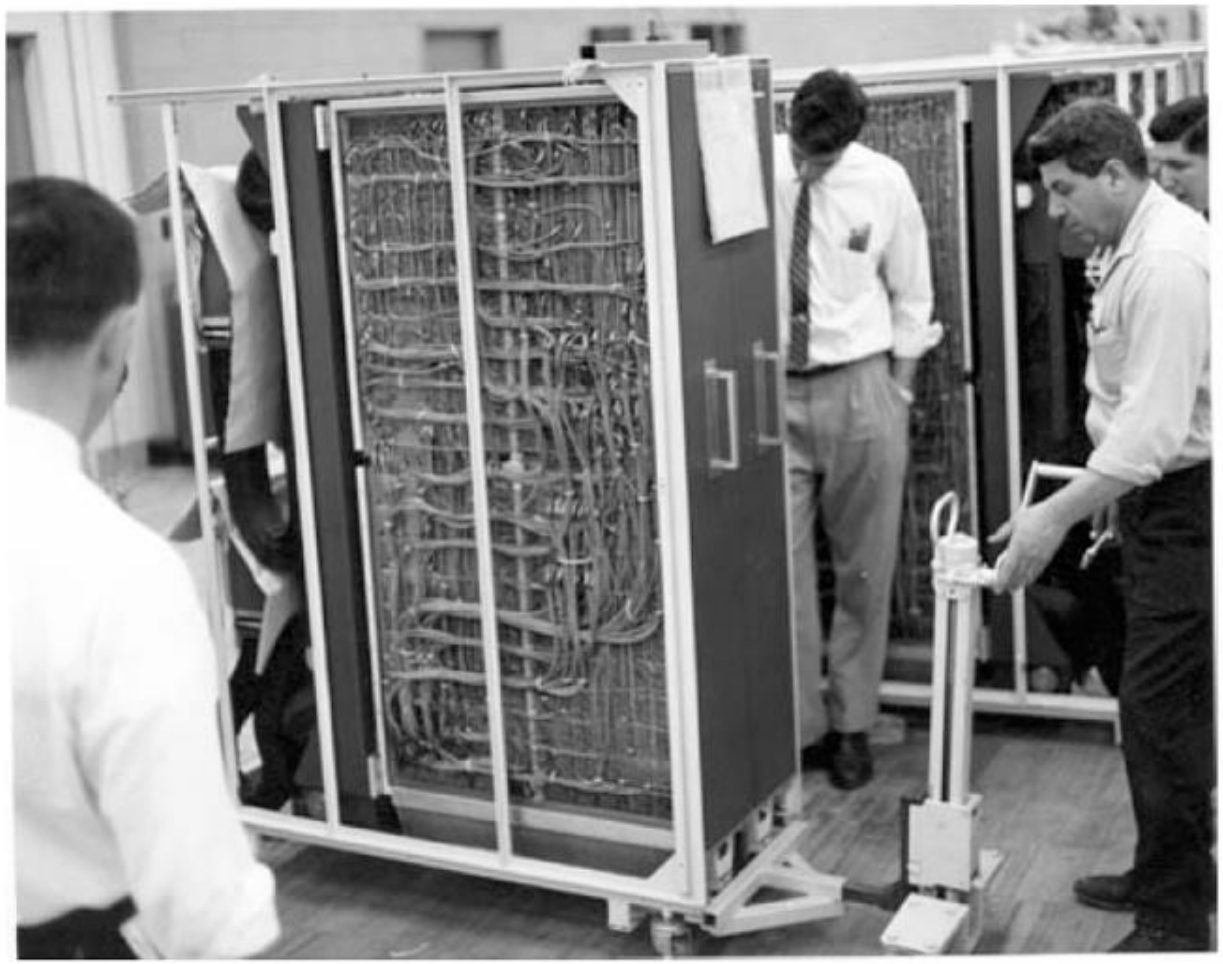
ALDs; each block was a logic gate and represented on a punched card. Machines did the routing and wiring



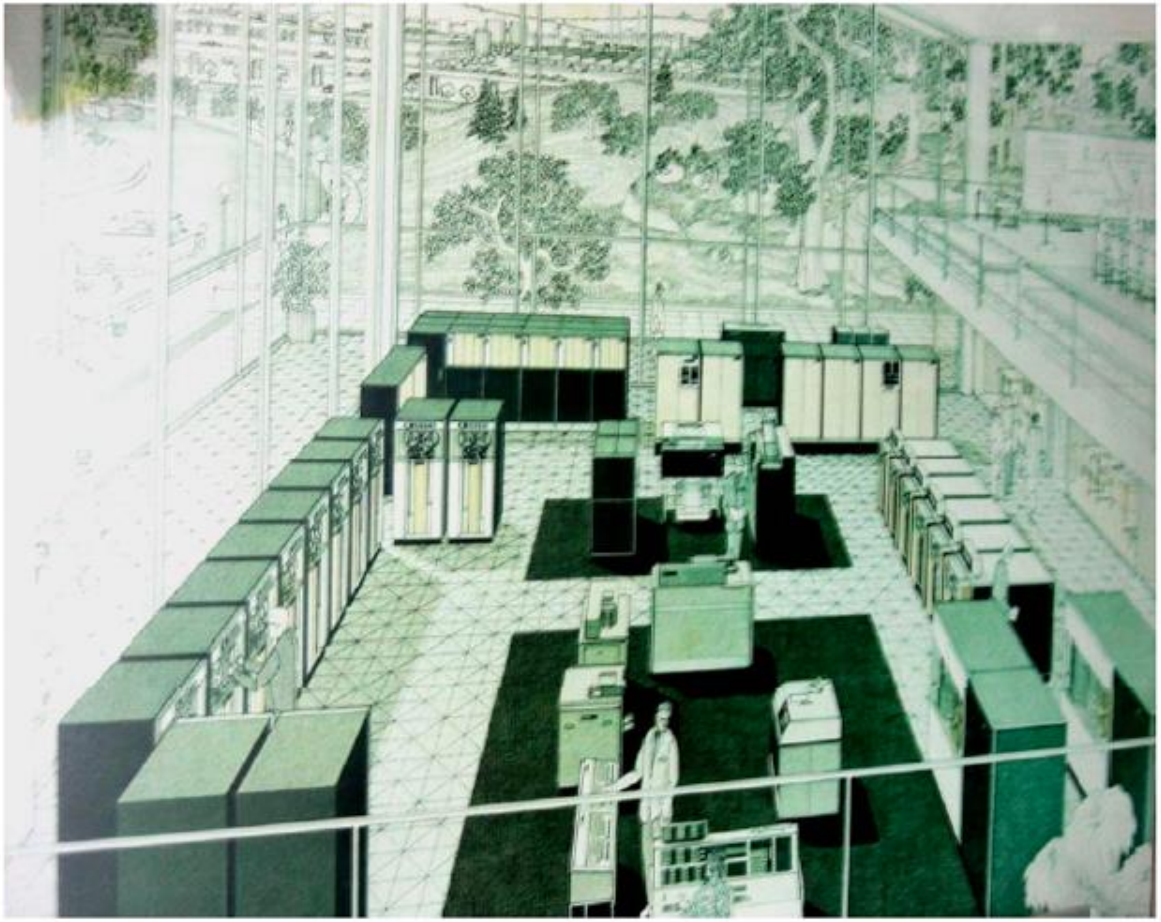
SMS Cards and Sockets



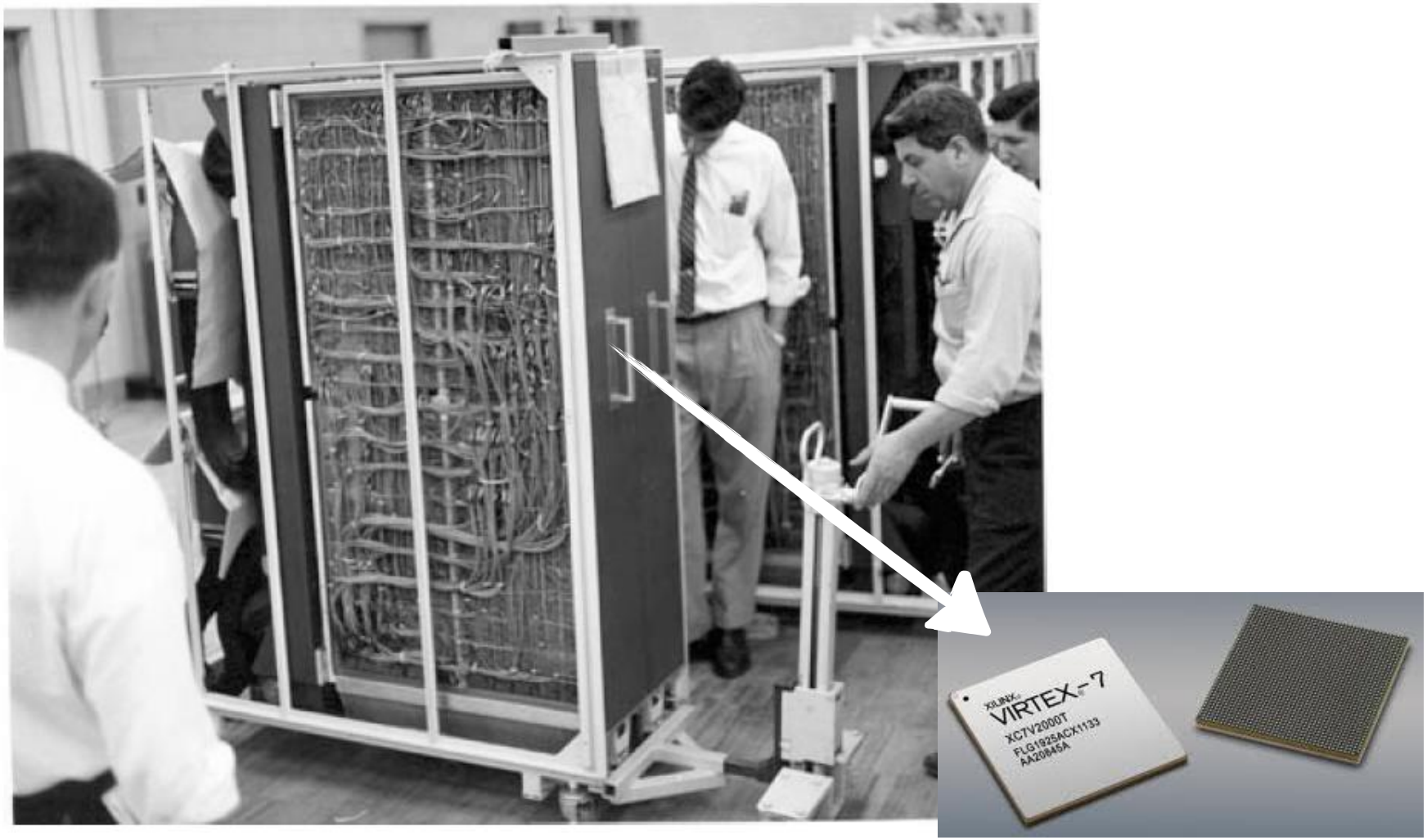
Large SMS frame (7030 and 7090)



7094 (c.1961); more than 350 made
enabled by design automation



50年的时间跨度



早期的可编程硬件

- PROM：可程序化只读存储器
- EPROM：紫外线可擦除只读存储器
- EEPROM：可擦写只读存储器
- 只能完成简单功能，可重复编程性差

FPGA的出现

- XC2064 1985
  - 第一个商用FPGA
  - 基于SRAM技术，可重复编程
  - 由Xilinx的两位创始人Ross Freeman和Bernard Vonderschmitt共同设计
  - Freeman因此进入美国发明家名人堂National Inventors Hall of Fame

FPGA技术的发展

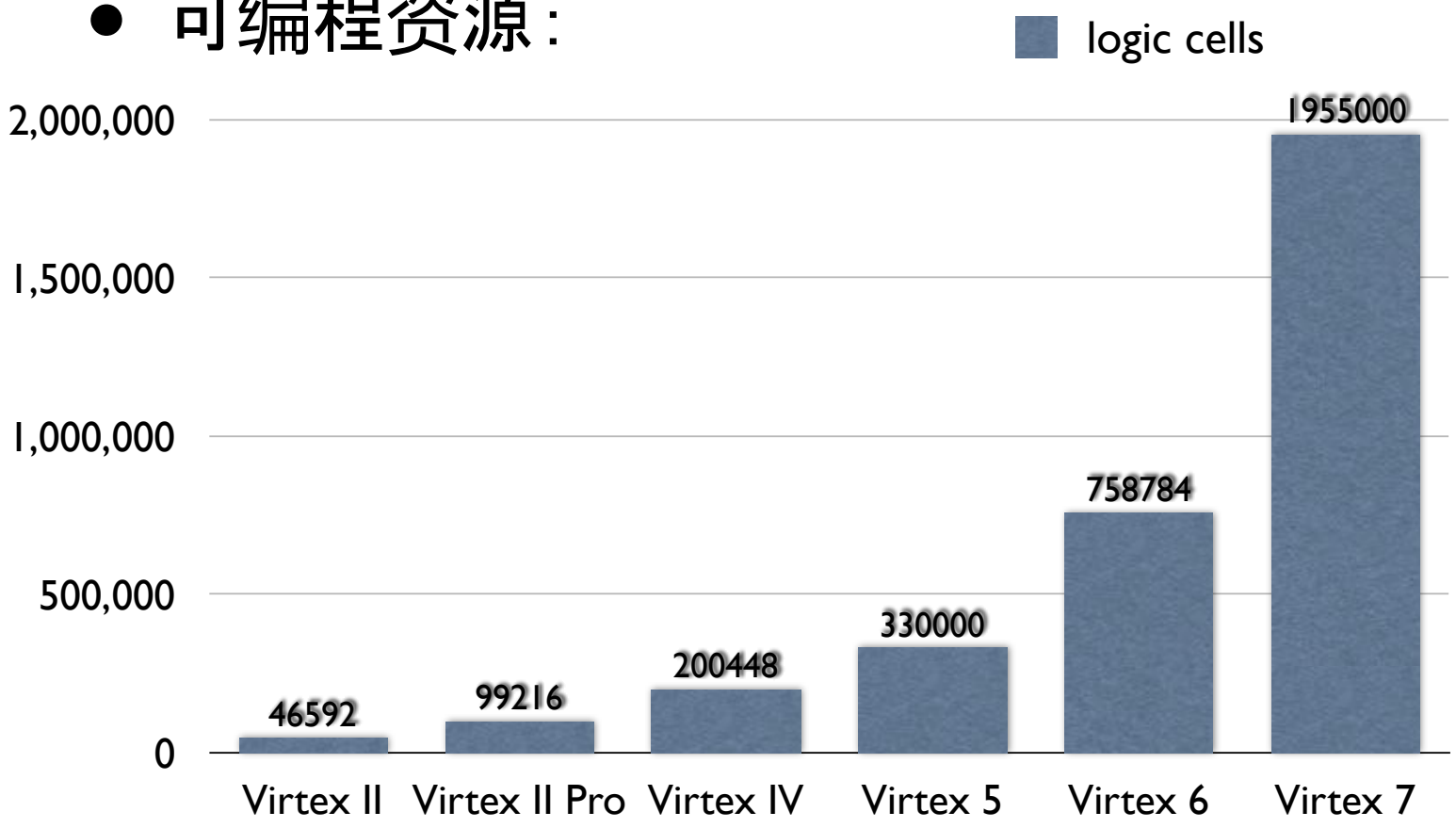
- 可编程基本单元：
  - 3-input lookup table（早期产品）
  - 4-input lookup table（90年代之后）
  - 6-input lookup table（Virtex-5之后）

FPGA技术的发展

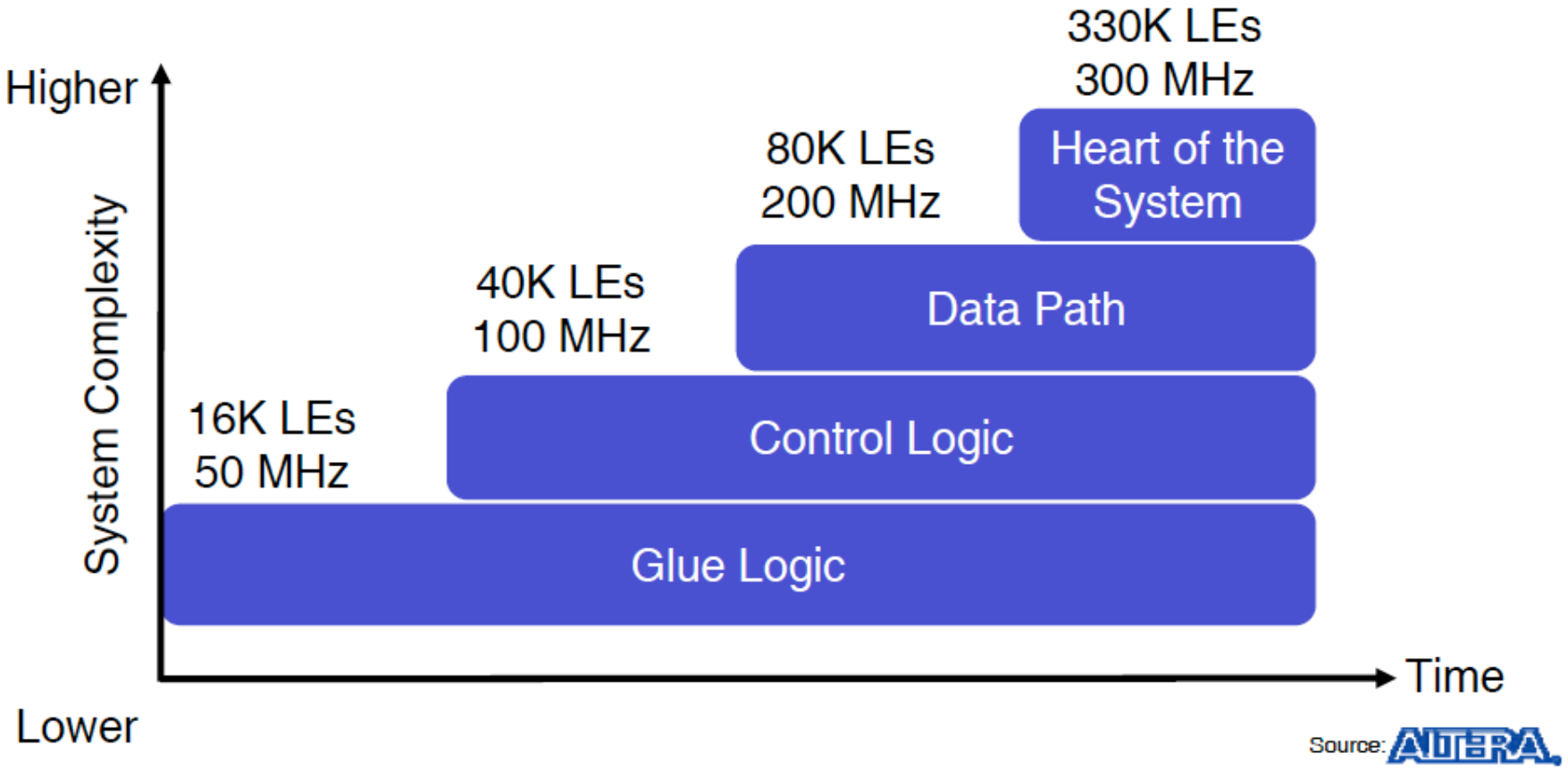
- 粗粒度单元的加入：
  - DSP48：25x18的乘法和累加
  - block RAM： 18-Kb or 36-Kb
  - ARM processor

FPGA技术的发展

- 可编程资源：

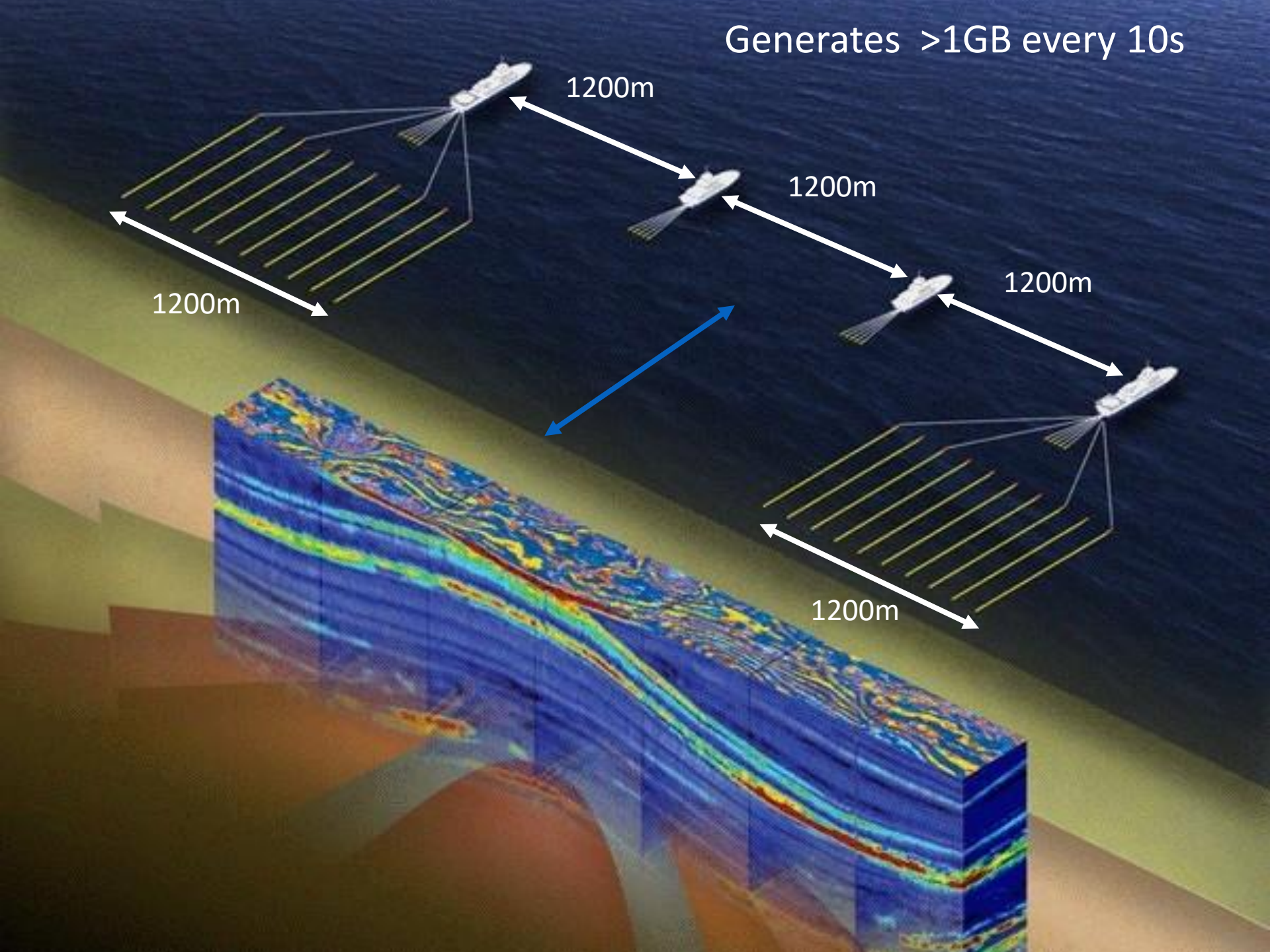


基于FPGA的应用发展



本讲提纲

- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限


Generates >1GB every 10s
1200m
1200m
1200m
1200m
1200m

Application Specific Number Representations

- Conventional computation platforms:
  - single/double precision floating-point
  - adapting precision is either not applicable or provides no significant performance benefit
- FPGAs: no restrictions on data format
  - support any kind of representation, such as fixed-point, floating-point, logarithmic number system (LNS) and residue number system (RNS)
  - bit-width of each variable is configurable
- Example: addition on Virtex IV FX100 FPGA
  - 32-bit floating-point: 100 adders
  - 16-bit fixed-point: 5000 adders

Precision V.S. Performance

- FPGAs enable application specific number representation
- We trade off precision (representation) with speed of computation
- How do we determine that the accelerated result is still good enough?

Evaluate the Accuracy of Seismic Images

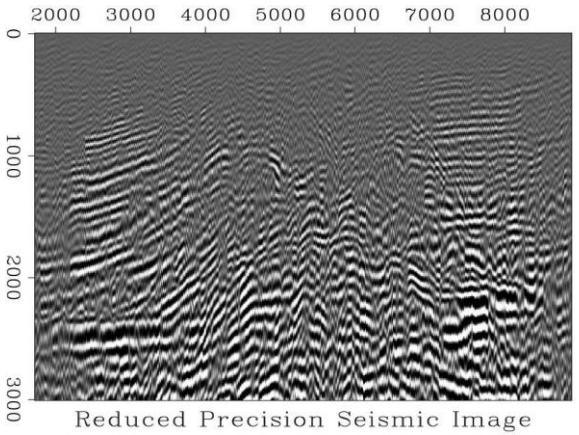
- Basic idea: compare the FPGA result with a ‘true’ image, i.e. the image calculated with single precision floating-point, which contains the correct image pattern

Evaluate the Accuracy of Seismic Images

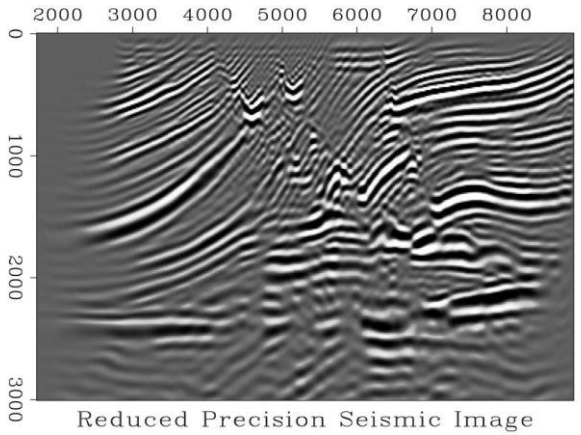
- We apply techniques based on Prediction Error Filters (PEFs) (Claerbout, J. 1994) to highlight the differences between ‘true’ image A and test image B
  - step 1, divide image A into overlapping small regions of 40x40 pixels, and estimate PEFs for the small regions
  - step 2, apply these PEFs to both image A and image B to get the results A’ and B’
  - step 3, apply algebraic combinations of the images A’ and B’ to acquire a value indicating difference, denoted as the Difference Indicator (DI) value

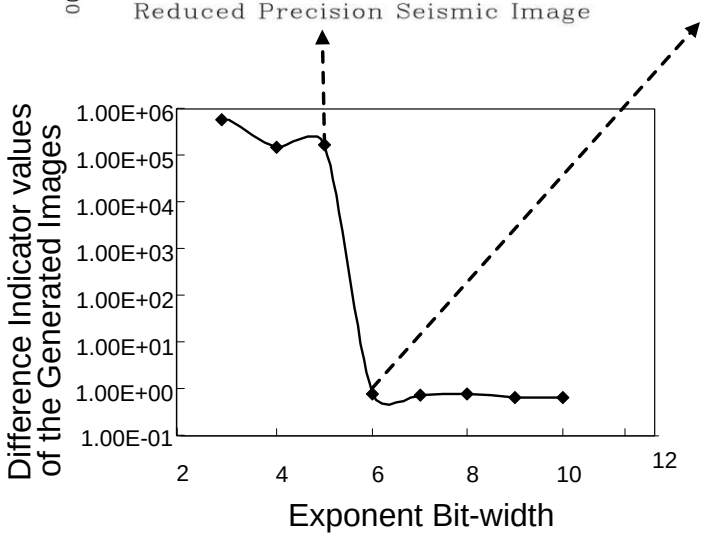
Exploration of Floating-point Designs

floating-point: 5-bit exponent

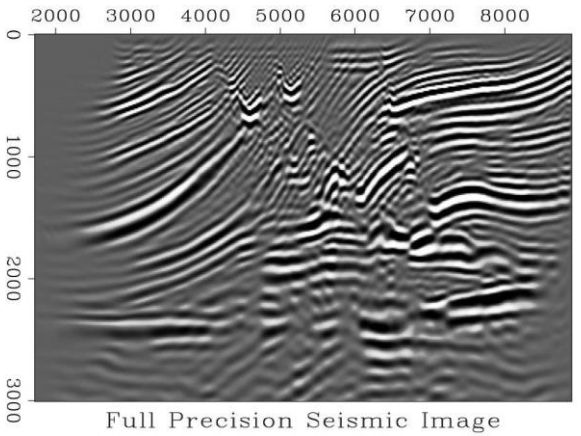


floating-point: 6-bit exponent



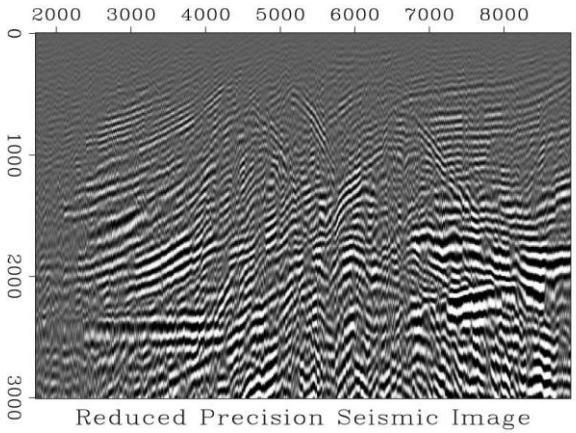


'true' image: single-precision floating-point

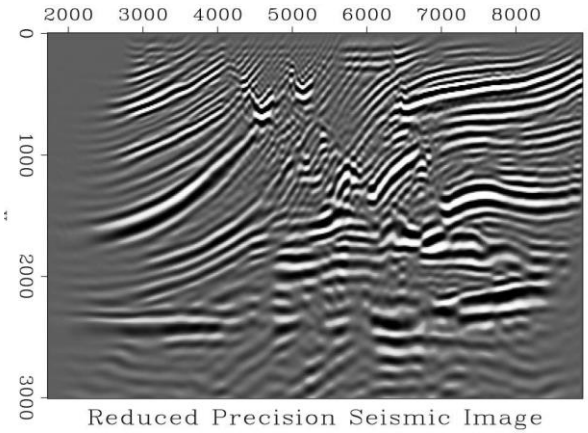


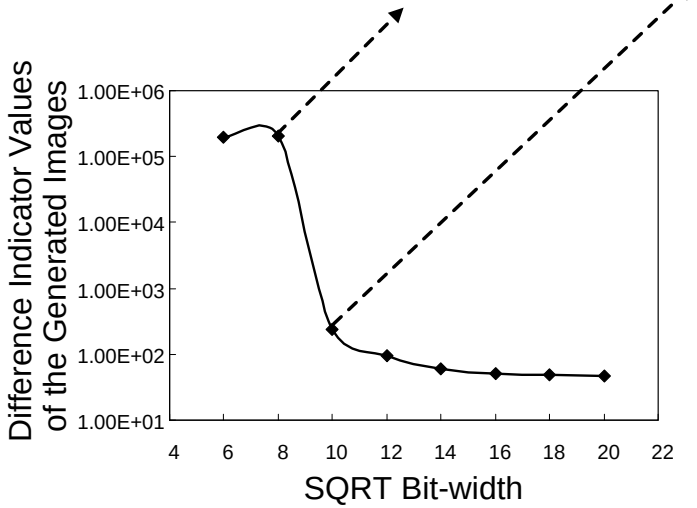
Exploration of Fixed-point Designs

8-bit fixed-point

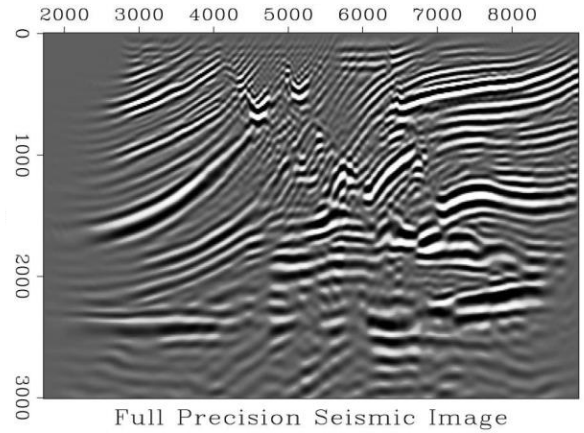


10-bit fixed-point





'true' image: single-precision floating-point



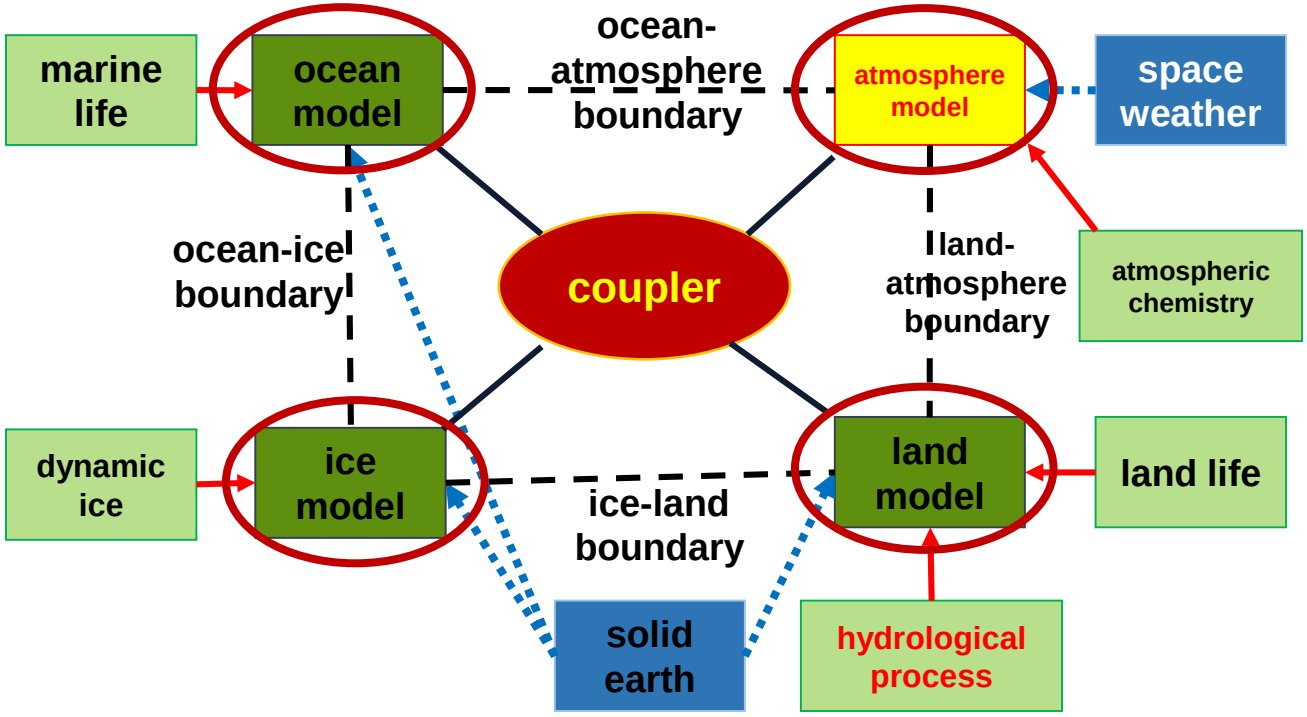
Complex Exponential

- 10-bit fixed-point numbers can achieve the same image quality as 32-bit floating-point numbers
- We can achieve 48x speedup using a reduced precision

Haohuan Fu, W. Osborne, Robert G. Clapp, Oskar Mencer, and Wayne Luk, “Accelerating Seismic Computations Using Customized Number Representations on FPGAs”, EURASIP Journal on Embedded Systems, vol. 2009, no. 1, 2009.

Demand for More Computing Power

- Increase in model complexity and resolution



Highly-Scalable Atmospheric Simulation Framework: Multidisciplinary Collaboration

- Prof. Chao Yang
  - Institute of Software, CAS
  - computational mathematics
- Dr. Wei Xue
  - Department of Computer Science, Tsinghua University
  - HPC (MPI, OpenMP, MIC)
- Dr. Haohuan Fu
  - Center for Earth System Science, Tsinghua University
  - HPC (accelerators, GPU, FPGA, MIC)
- Prof. Lanning Wang
  - College of Global Change and Earth System Science, Beijing Normal University (BNU-ESM)
  - climate scientist

Highly-Scalable Atmospheric Simulation on Tianhe-1A

- how to utilize peta-scale computing power?
- how to utilize hybrid computation resources?

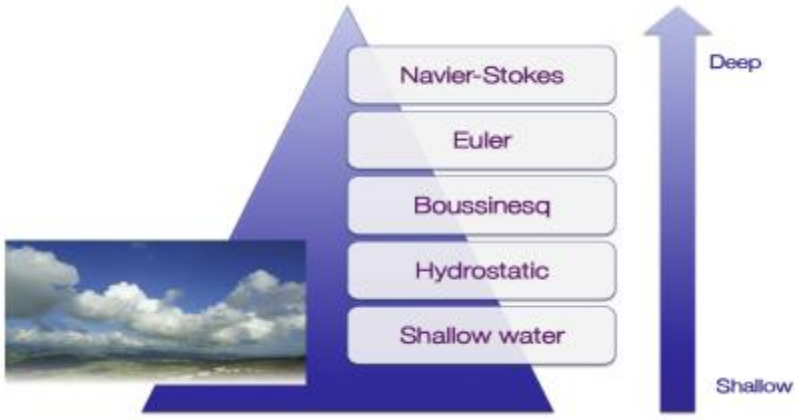
- Tianhe-1A
  - 2.6 Pflops, top1: Nov/2010
  - fast proprietary network
  - 7168 computing nodes
  - in each computing node
    - two 6-core Intel X5670 CPUs: 12 cores, 140 Gflops
    - one NVIDIA M2050 GPU: 14 streaming multiprocessors (448 CUDA cores), 515 Gflops

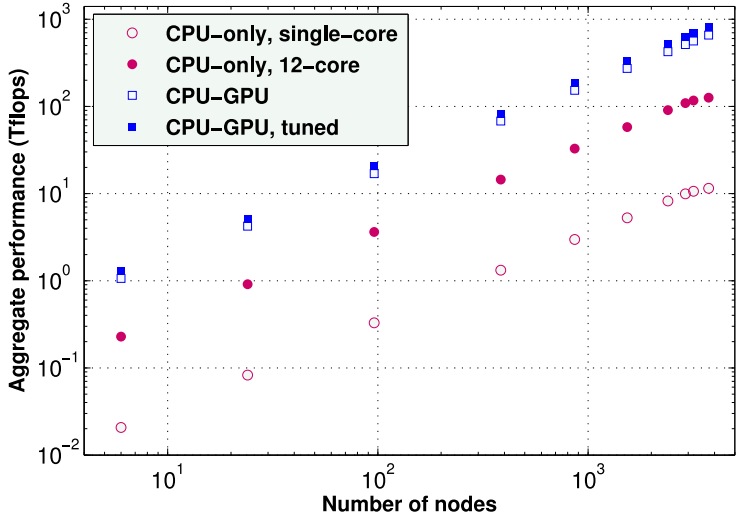
NSCC-Tianjin

The Tianhe-1A Supercomputer

Highly-Scalable Framework for Global Atmospheric Simulation on Tianhe-1A

- Starting from shallow wave equation
  - cubed-sphere mesh grid
  - adjustable partition between CPU and GPU
  - scale to 40,000 CPU cores and 3750 GPUs with a sustainable performance of 800 TFlops





For more details, please refer to our PPoPP 2013 paper: “A Peta-Scalable CPU-GPU Algorithm for Global Atmospheric Simulations”, in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pp. 1-12, Shenzhen, 2013. .

Highly-Scalable Atmospheric Simulation on Tianhe-2

- Tianhe-2 (the current top1)
  - LINPACK performance: 33.8 Pflops
  - 16,000 nodes
  - each node contains:
    - two 12-core Intel Ivy Bridge CPU, 422 Gflops
    - three Xeon Phi MIC cards (61 x86 cores in each card), 3 Tflops

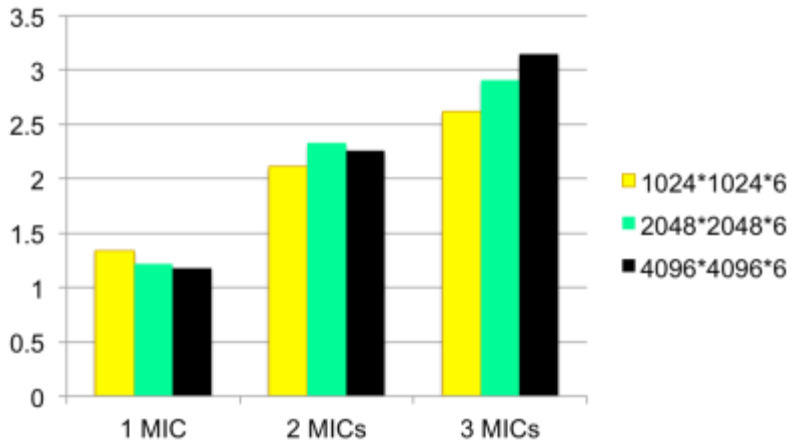
Highly-Scalable Framework for Global Atmospheric Simulation on Tianhe-2

From Tianhe-1A to Tianhe-2

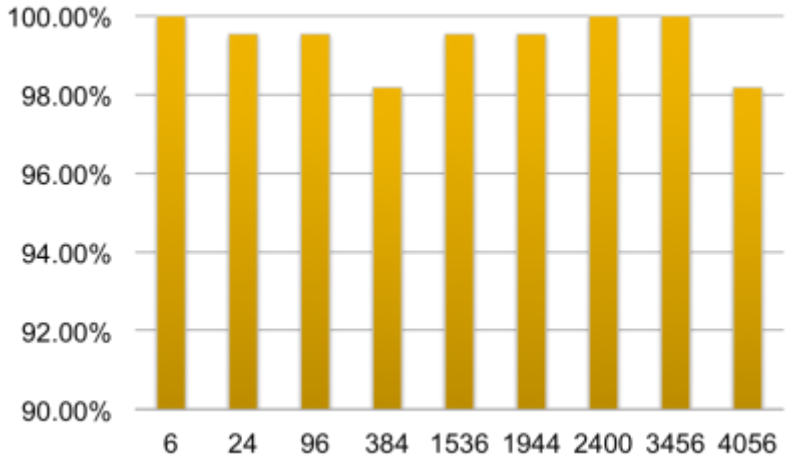
- 7,168 nodes to 16,000 nodes
- 2.6 Pflops to 33.9 Pflops
- “CPU+GPU” to “CPU+MIC”

Balanced utilization of CPU and MIC

- supporting 0 to 3 MICs
- overlapping of computation and communication



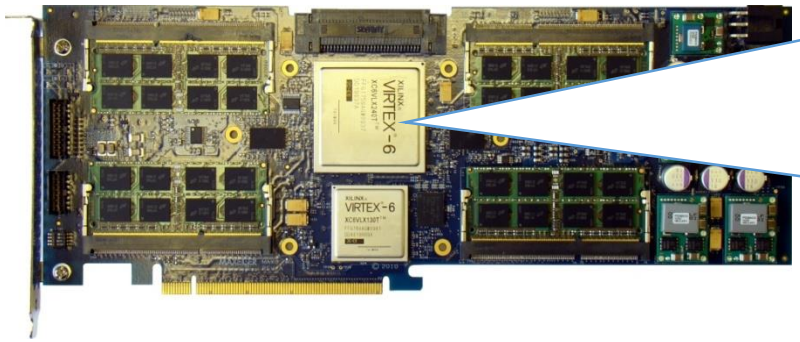
speedup: MIC over 24 CPU cores

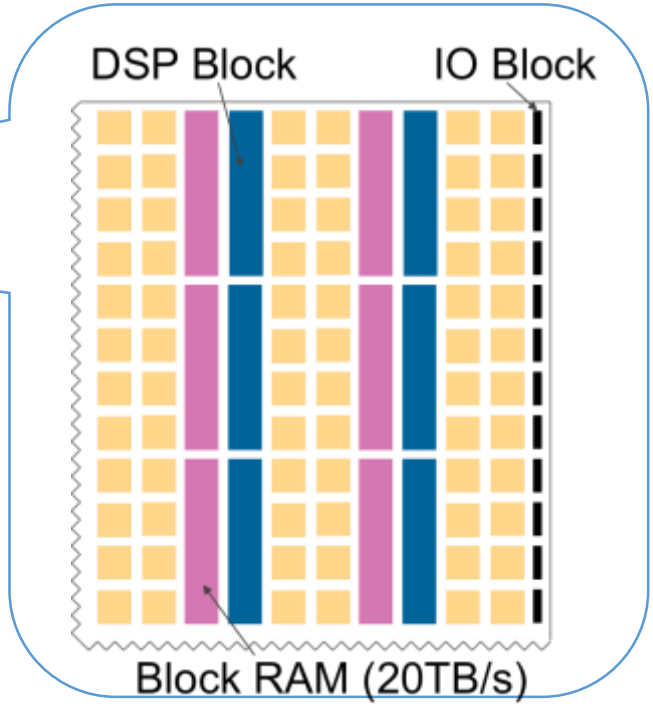


scalability

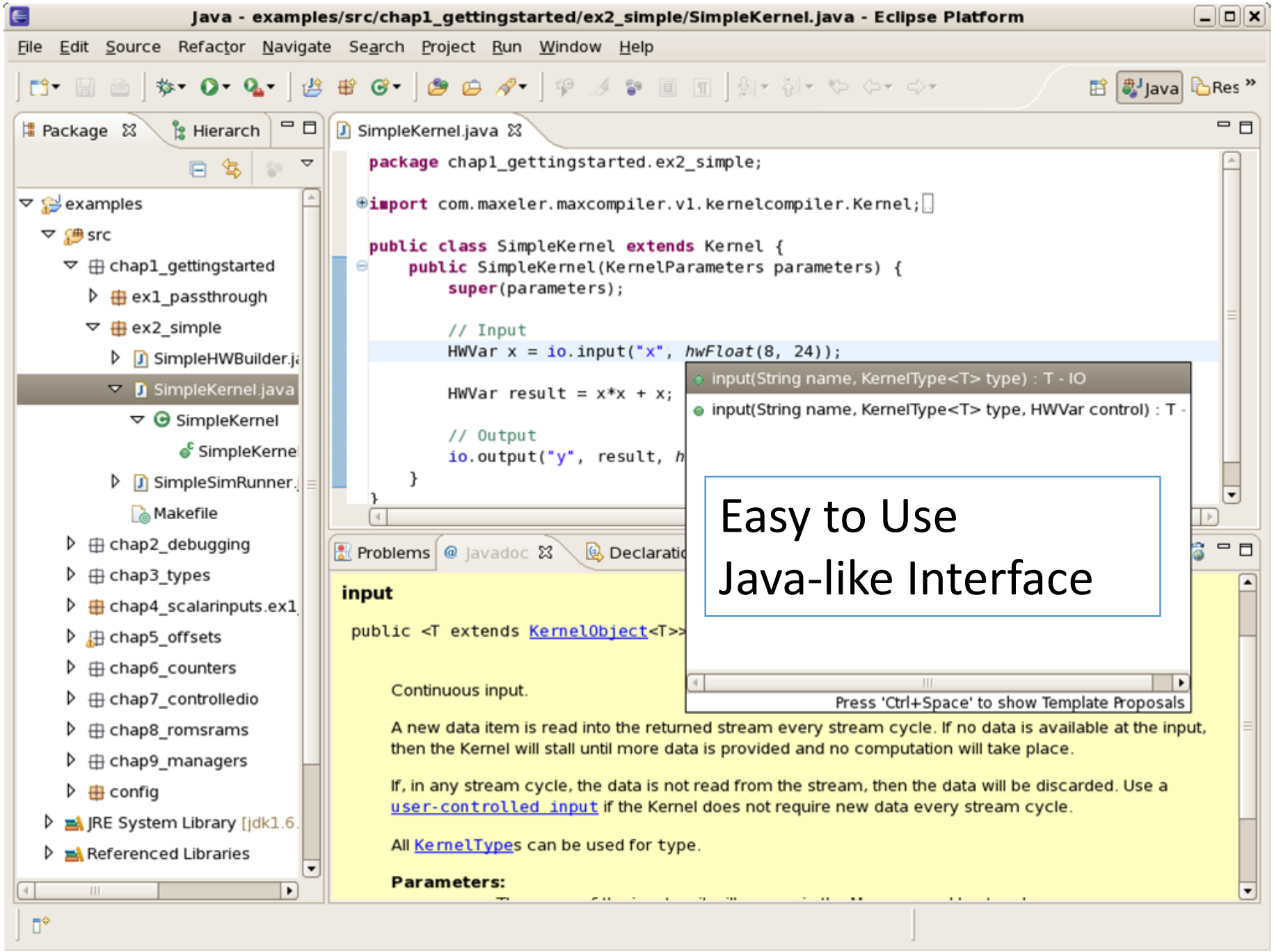
Highly-Scalable Atmospheric Simulation on Data-Flow Engines

- Maxeler Data-Flow Engine (DFE)
  - Field-Programmable Gate Arrays (FPGA)
  - 24 GB onboard memory
  - PCIE connection to host
  - MaxRing connection between cards



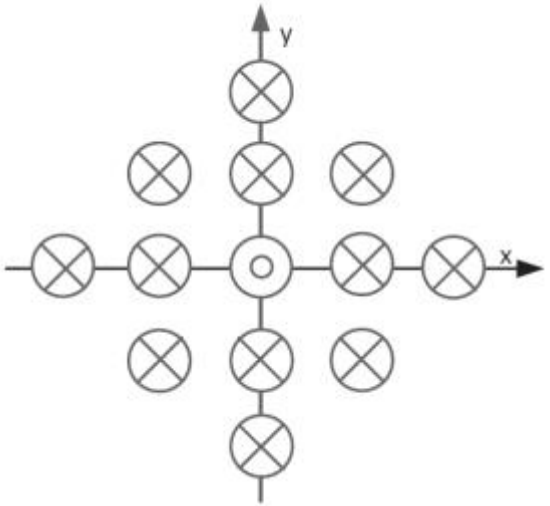


A Practical
Data Flow Design
with 4866 nodes

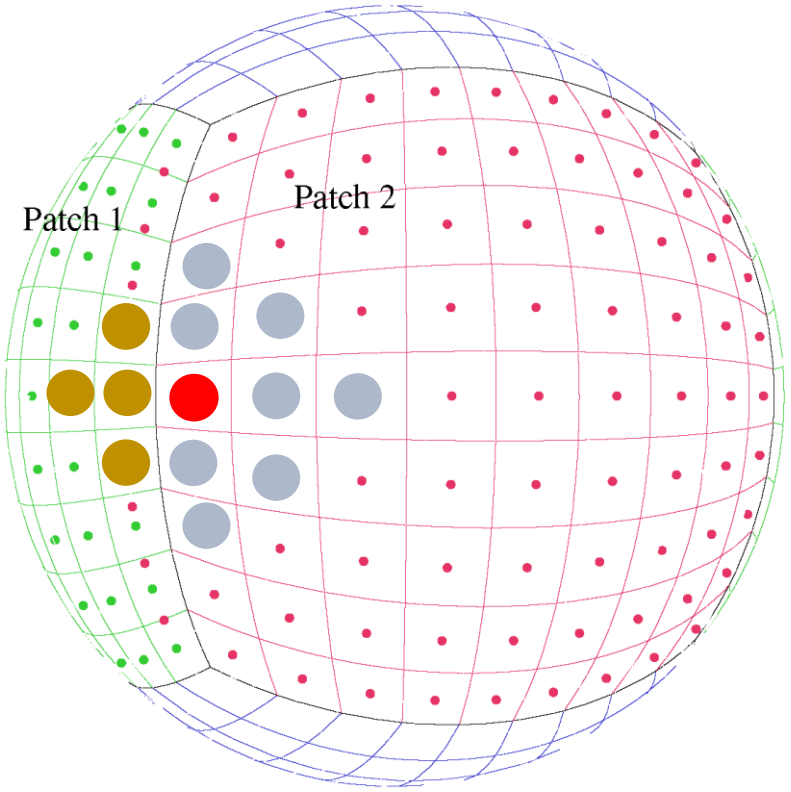

Java - examples/src/chap1_gettingstarted/ex2_simple/SimpleKernel.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Package Hierarch
examples
src
chap1_gettingstarted
ex1_passthrough
ex2_simple
SimpleHWBuilder.j
SimpleKernel.java
SimpleKernel
SimpleKerne
SimpleSimRunner.
Makefile
chap2_debugging
chap3_types
chap4_scalarinputs.ex1
chap5_offsets
chap6_counters
chap7_controlledio
chap8_romsrams
chap9_managers
config
JRE System Library [jdk1.6.
Referenced Libraries
SimpleKernel.java
package chap1_gettingstarted.ex2_simple;
import com.maxeler.maxcompiler.v1.kernelcompiler.Kernel;
public class SimpleKernel extends Kernel {
public SimpleKernel(KernelParameters parameters) {
super(parameters);
// Input
HWVar x = io.input("x", hwFloat(8, 24));
HWVar result = x*x + x;
// Output
io.output("y", result, h
}
}
input(String name, KernelType<T> type) : T - IO
input(String name, KernelType<T> type, HWVar control) : T -
Press 'Ctrl+Space' to show Template Proposals
Easy to Use
Java-like Interface
Problems Javadoc Declaratio
input
public <T extends KernelObject<T>>
Continuous input.
A new data item is read into the returned stream every stream cycle. If no data is available at the input, then the Kernel will stall until more data is provided and no computation will take place.
If, in any stream cycle, the data is not read from the stream, then the data will be discarded. Use a user-controlled input if the Kernel does not require new data every stream cycle.
All KernelTypes can be used for type.
Parameters:

Mesh and Stencil

- 13-point stencil



- Interp. Across patches
  - 1-d linear interpolation



- Spatially discretized with a cell-centred finite volume method
- Integrated with a second-order accurate TVD Runge-Kutta method

Hybrid CPU+FPGA Design

For each stencil cycle

FPGA side:

①Inner-part stencil

CPU side:

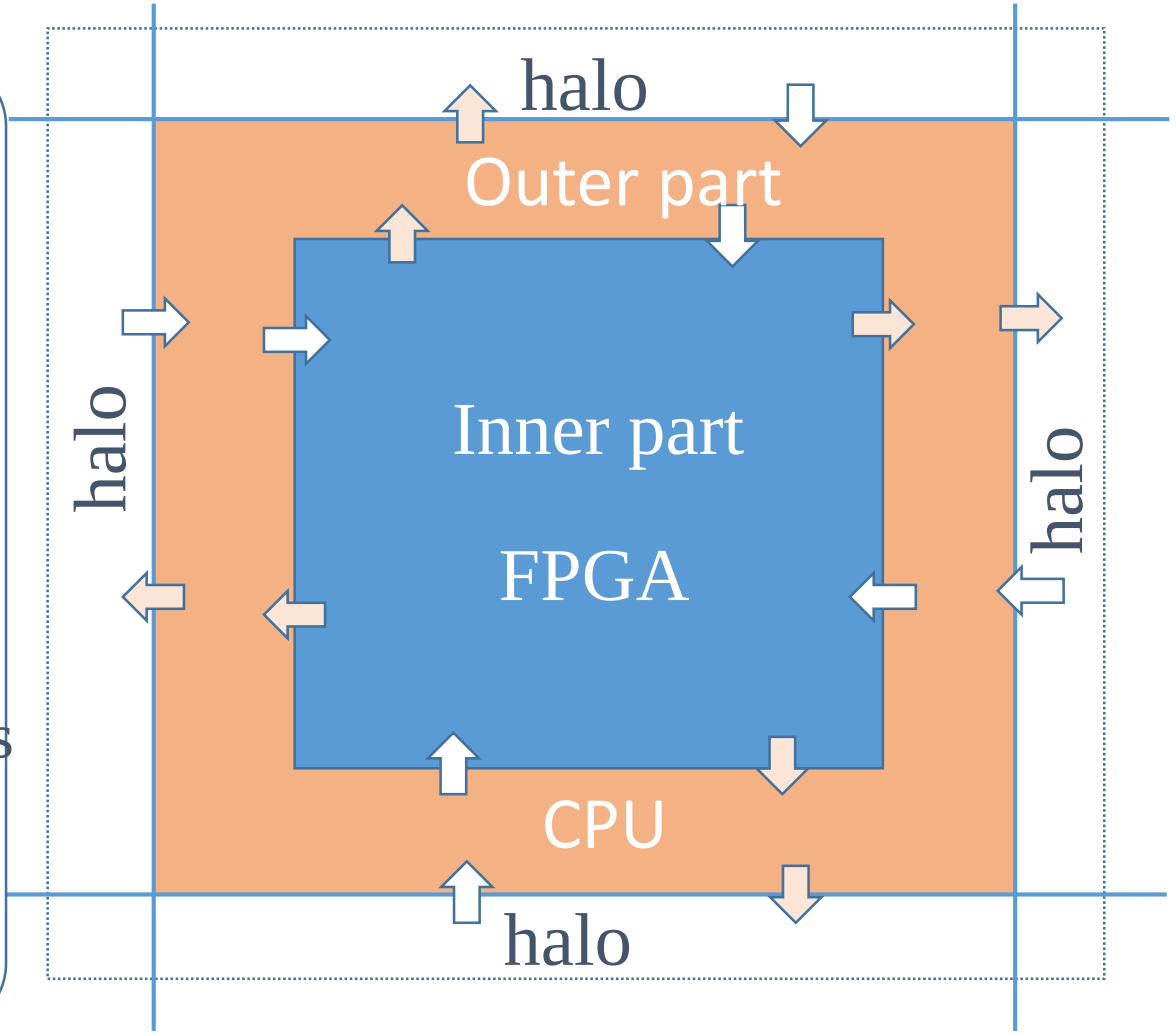
①Update halos

②Interpolate if necessary

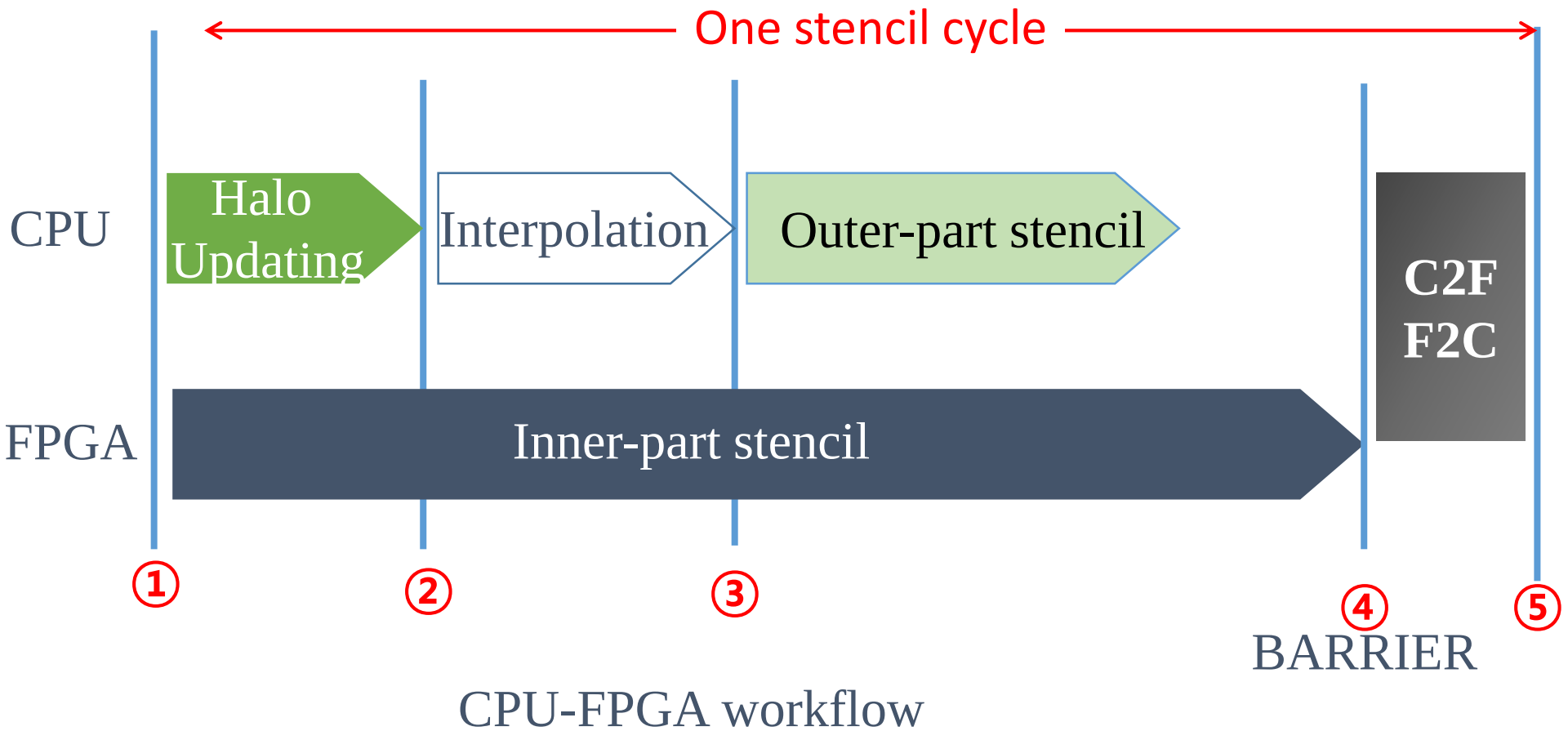
③Outer-part stencils

BARRIER:

CPU-FPGA exchange



Work Flow



CPU-FPGA workflow

Go for a Mixed-Precision Design

Floating point operations of SWE stencil

Operations	num
ADD/SUB	434
MUL	570
DIV	99
Others	45

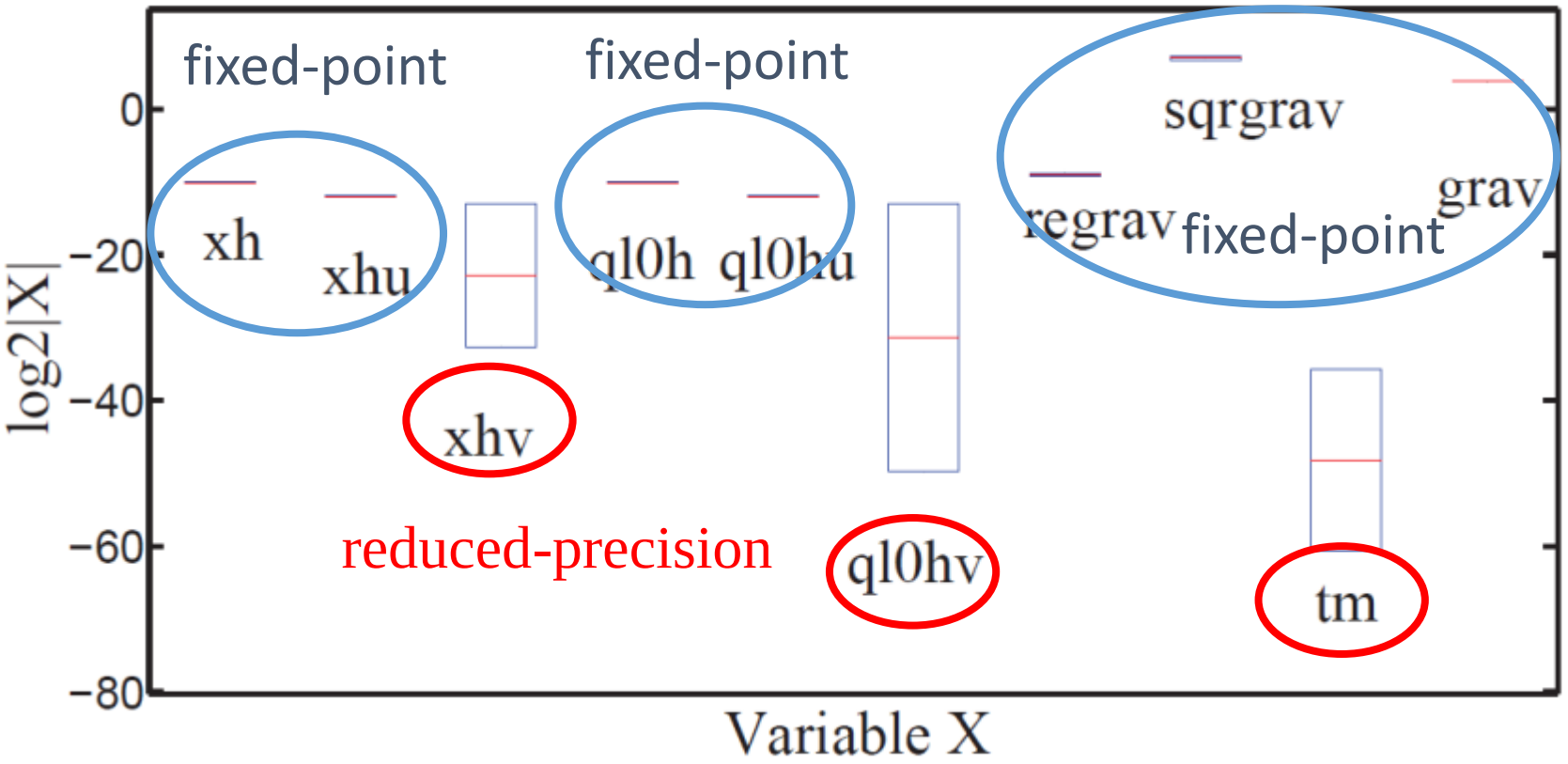
Resource Cost of SWEs on Virtex-6 SX457T

Resource	baseline
LUTs	299 %
FFs	220 %
BRAMs	20 %
DSPs	189 %

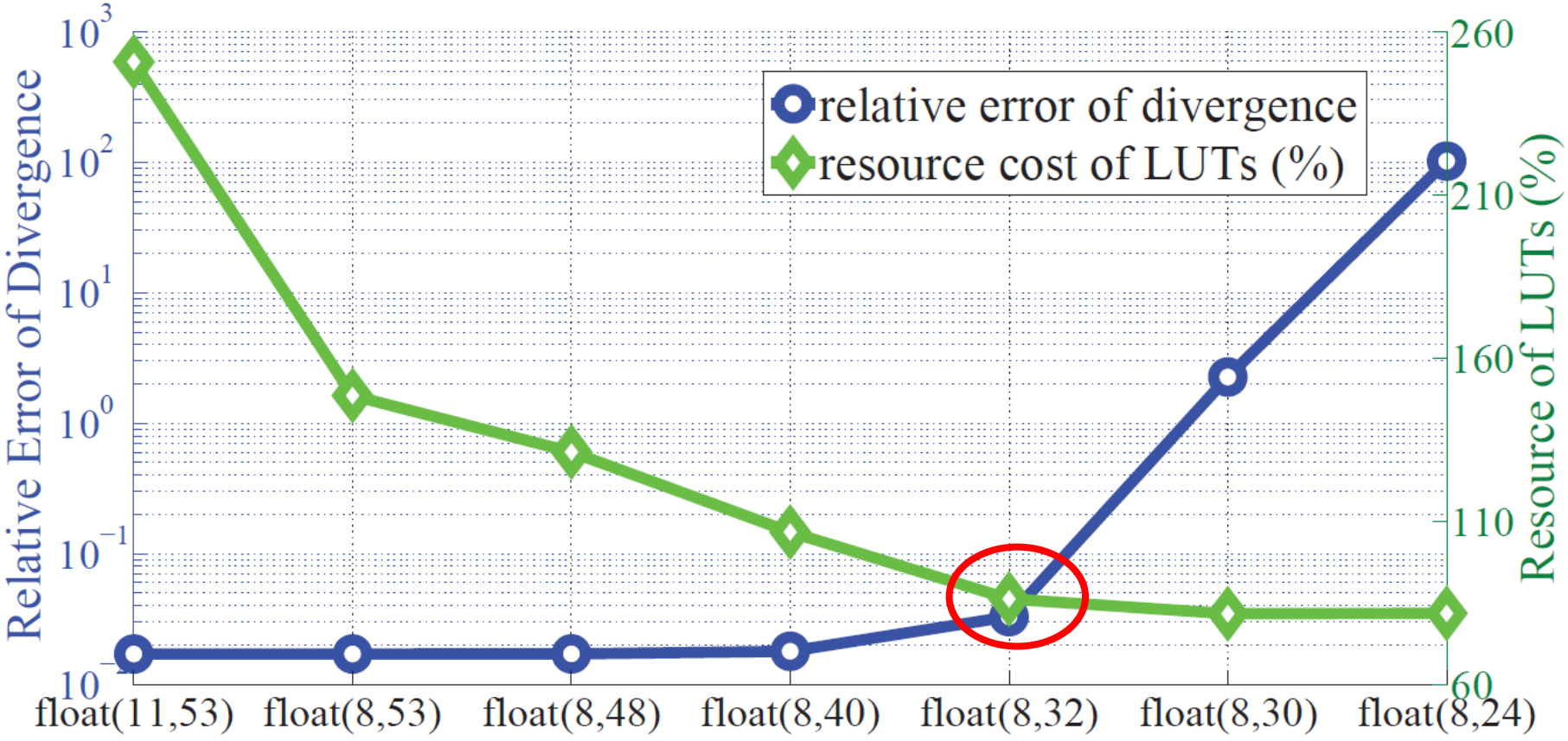
Baseline: a straightforward double-precision SWEs

- Precision-based optimization to further decrease the resource usage

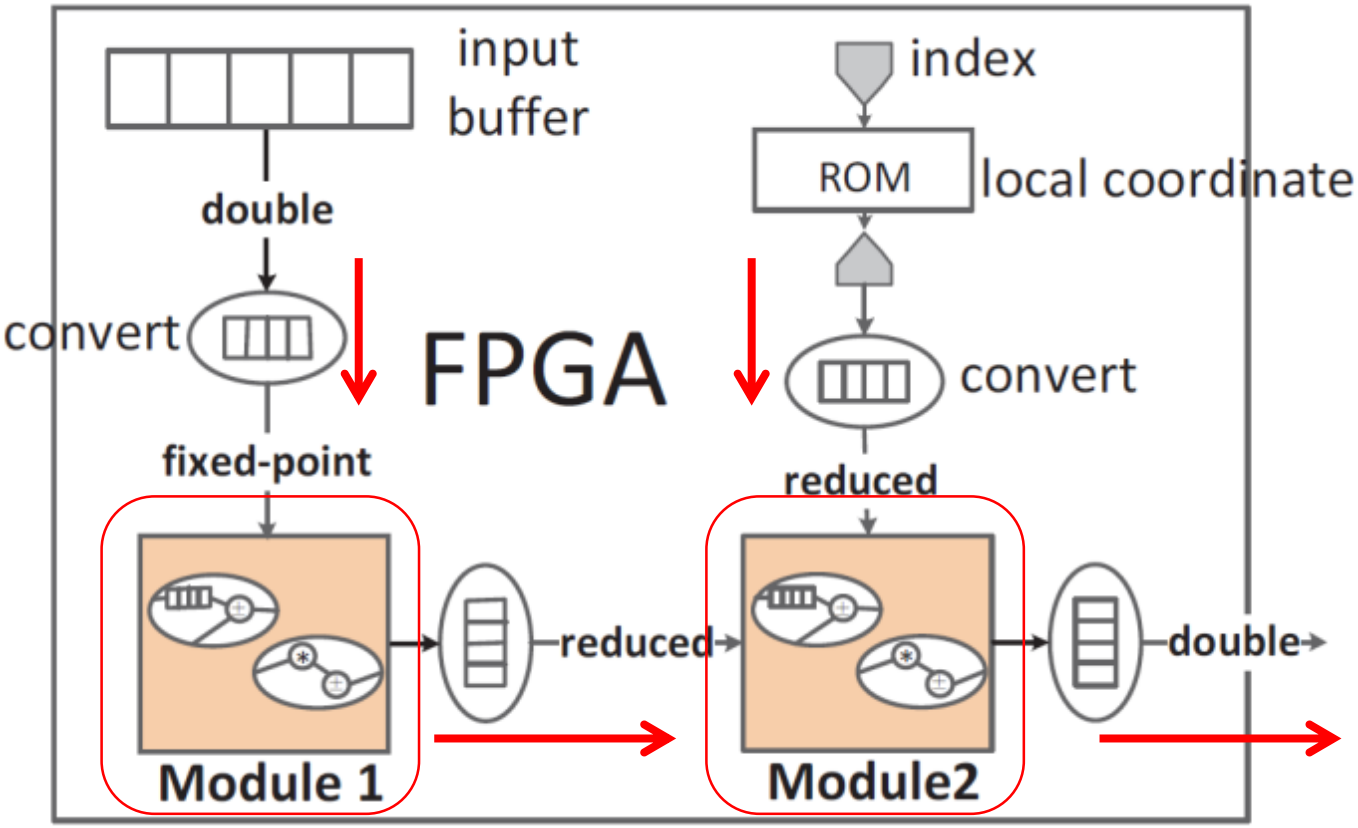
Analysis of the Dynamic Range



Precision Exploration



General Architecture of the Mixed-Precision Design



Resource Cost of SWEs on Virtex-6 SX457T

Resource	baseline	mixed-precision
LUTs	299 %	76.17%
FFs	220 %	53.41%
BRAMs	20 %	12.59 %
DSPs	189 %	44.84 %

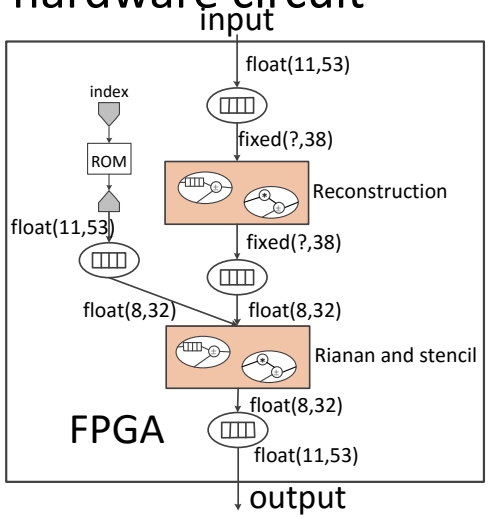
- Baseline: a straightforward double-precision SWEs
- Mixed-precision: fixed-point and reduced-precision floating-point

Highly-Scalable Framework for Global Atmospheric Simulation on Data-Flow Engines

- Converting a software program into a customized hardware circuit

Algorithm 1 SWEs Algorithm

```
1: for patch 0 to patch 5 do
2:     Halo Updating
3:     for j ← 0 to n_j do
4:         for i ← 0 to n_i do
5:             Compute Local Coordinate                          ▷ 58 FLOP
6:             Compute Left Intermediate Value, Including{
7:                 State Reconstruction                          ▷ 218 FLOP
8:                 if i==0 then
9:                     Left Boundary computing 1                 ▷ 400 FLOP
10:                elsif i==1 then
11:                    Left Boundary computing 2                 ▷ 388 FLOP
12:                endif
13:                Rianann Operations }                          ▷ 67 FLOP
14:            Compute Right Intermediate Value Including the Right Boundary
15:            Compute Bottom Intermediate Value Including the Bottom Boundary
16:            Compute Top Intermediate Value Including the Top Boundary
17:            Compute Upwind Stencil, h, hu^1 and hu^2          ▷ 810 FLOP
18:        end for
19:    end for
20: end for
```



Mesh size: $1024 \times 1024 \times 6$					
platform	performance (points/second)	speedup	power (Watt)	efficiency (points/(second·Watt))	power efficiency
6-core CPU	4.66K	1	225	20.71	1
Tianhe-1A node	110.38K	23x	360	306.6	14.8x
MaxWorkstation	468.11K	**100x**	186	2.52K	**121.6x**
MaxNode	1.54M	**330x**	514	3K	**144.9x**

For more details, please refer to our FPL 2013 paper: "Accelerating Solvers for Global Atmospheric Equations Through Mixed-Precision Data Flow Engine", in *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications*, 2013.

本讲提纲

- 什么是data flow computing
- 现有的平台及编程工具
- 发展历史
- 应用举例
- 讨论：优势与局限

优势与局限

什么是适合应用的并行化

“如果你需要开垦一片土地，你会使用什么?”

Seymour Cray (ca 1970)

什么是适合应用的并行化

“如果你需要开垦一片土地，你会使用什么？
两只强壮的牛 还是 1024只鸡？”

Seymour Cray (ca 1970)

可重构的适合应用的并行化

“如果你需要开垦一片土地，你会使用什么？
两只强壮的牛 还是 1024只鸡？”

Seymour Cray (ca 1970)

使用可重构计算，我们可以同时提供两种解决方案！

- 用两只牛来耕地，然后可重构为
- 1024只鸡来下蛋，吃虫子…

可重构的适合应用的并行化

“如果你需要开垦一片土地，你会使用什么？
两只强壮的牛 还是 1024只鸡？”

Seymour Cray (ca 1970)

使用可重构计算，我们可以同时提供两种解决方案！

- 用两只牛来耕地，然后可重构为
- 1024只鸡来下蛋，吃虫子…

可重构计算：有效，有趣，变不可能为可能