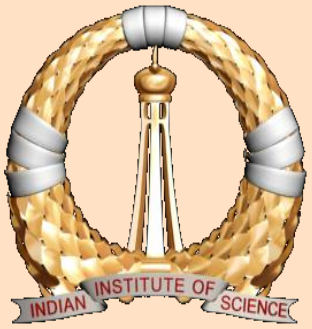
Programmability, Portability and Performance in Heterogeneous Accelerator-Based Systems

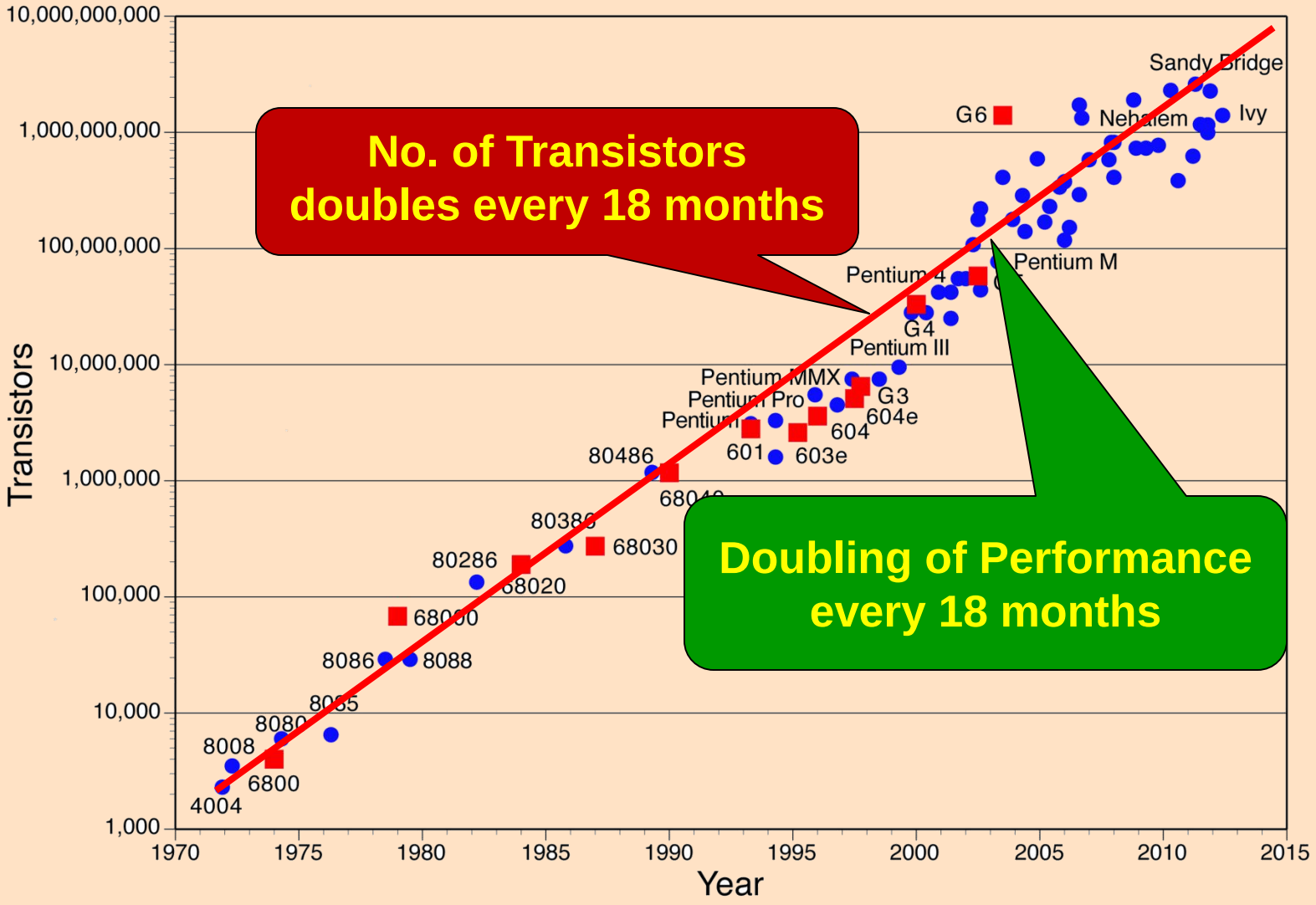
R. Govindarajan
High Performance Computing Lab.
SERC & CSA, IISc
govind@serc.iisc.ernet.in



Overview

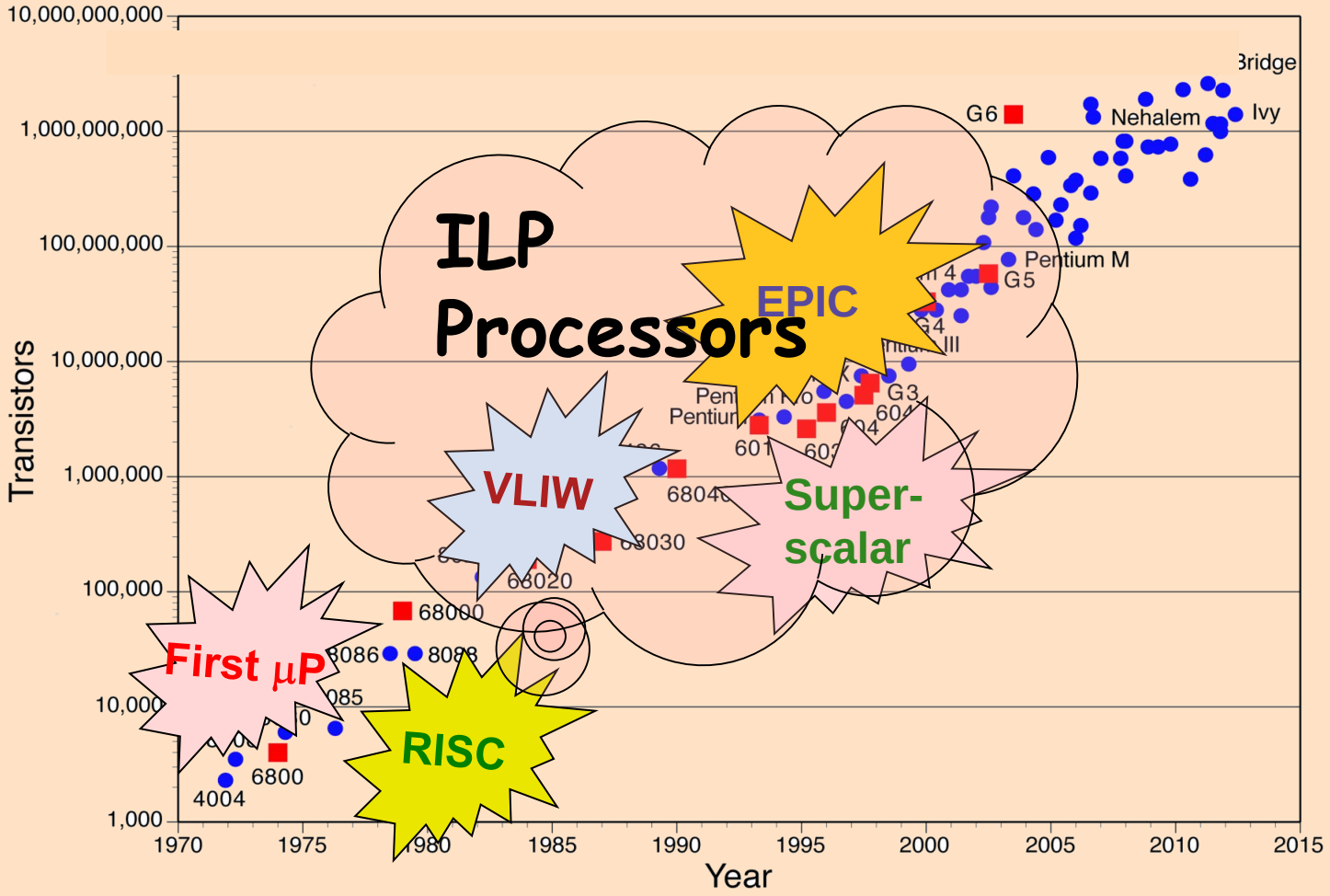
- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

Moore's Law

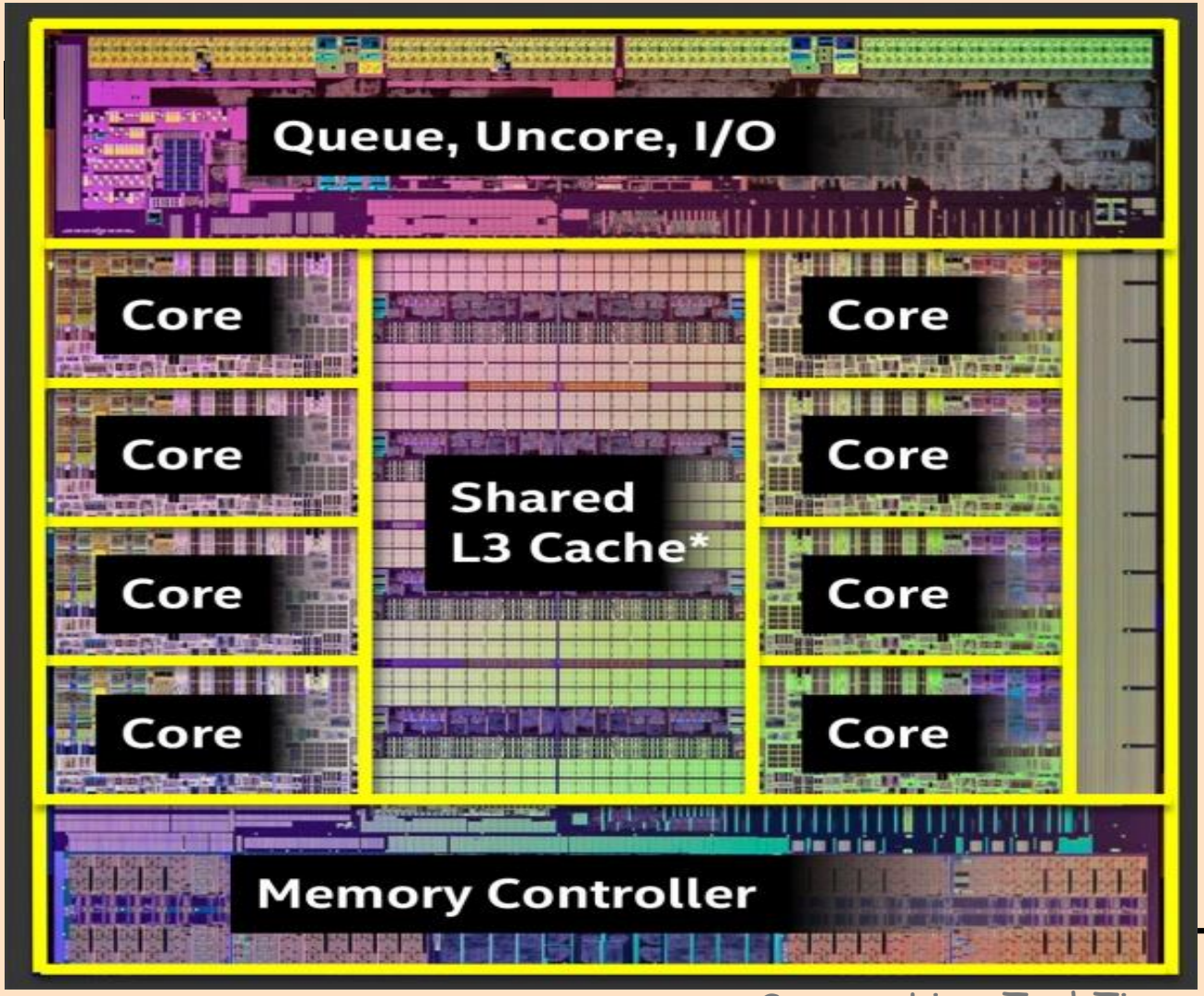


Source: Univ. of Wisconsin

Moore's Law: Processor Architecture Roadmap



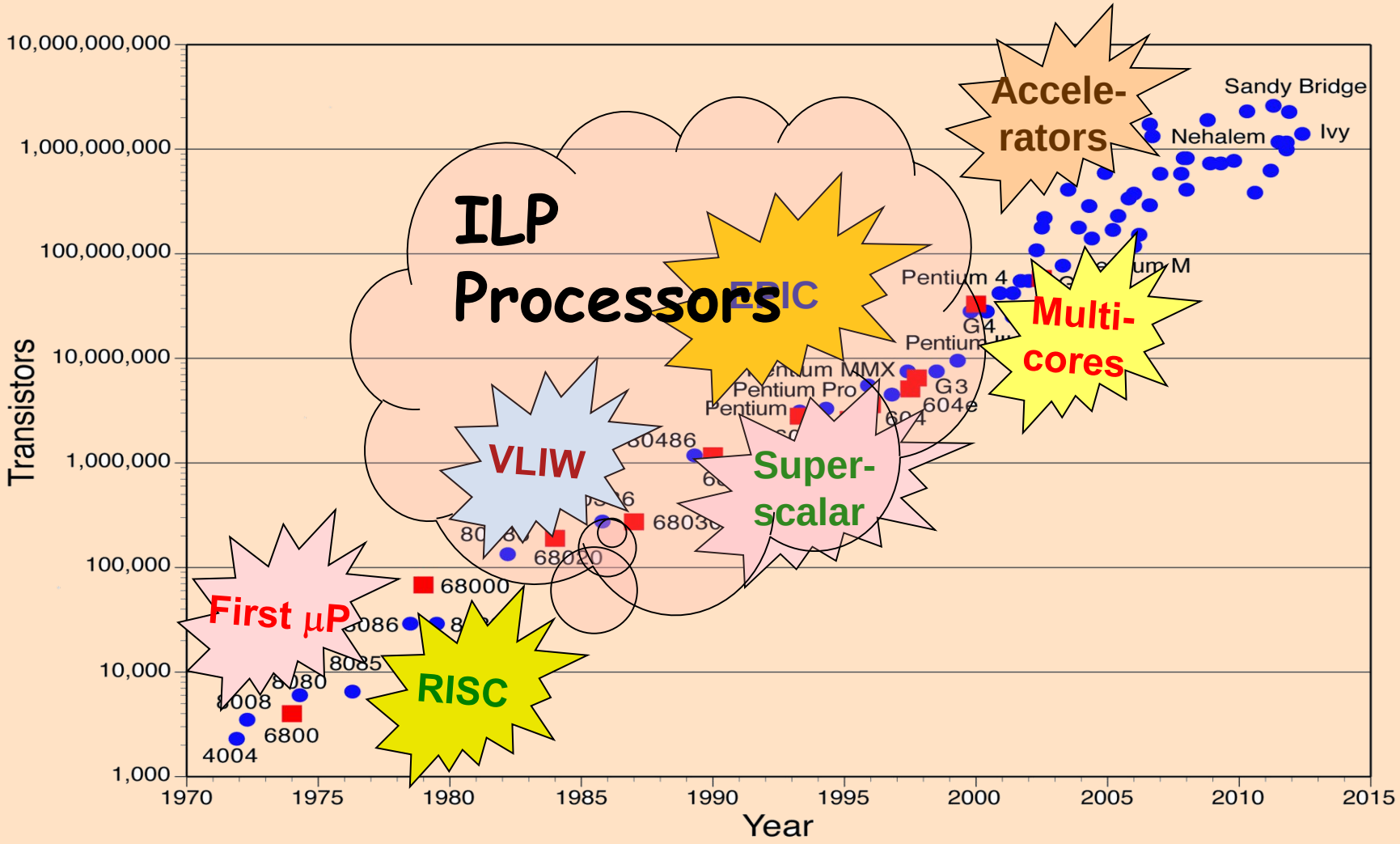
Multicores : The "Right Turn"



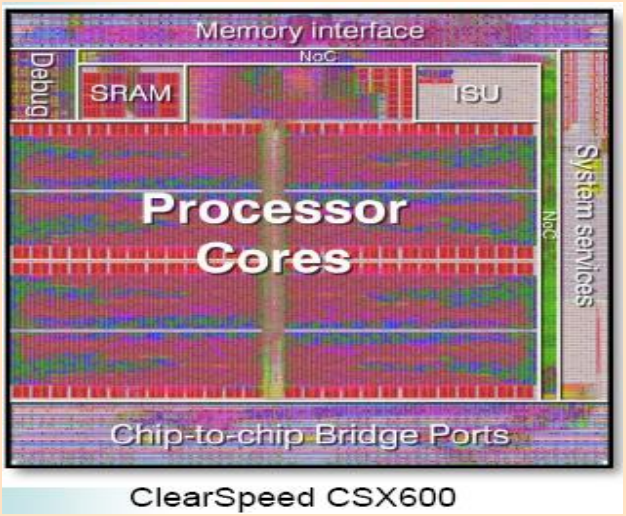
Performance

Source: LinusTechTips

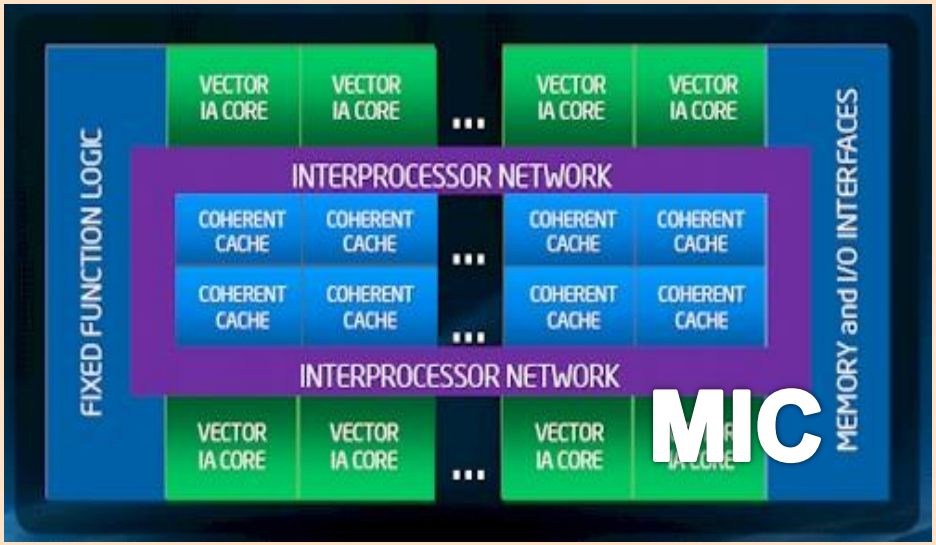
Moore's Law: Processor Architecture Roadmap

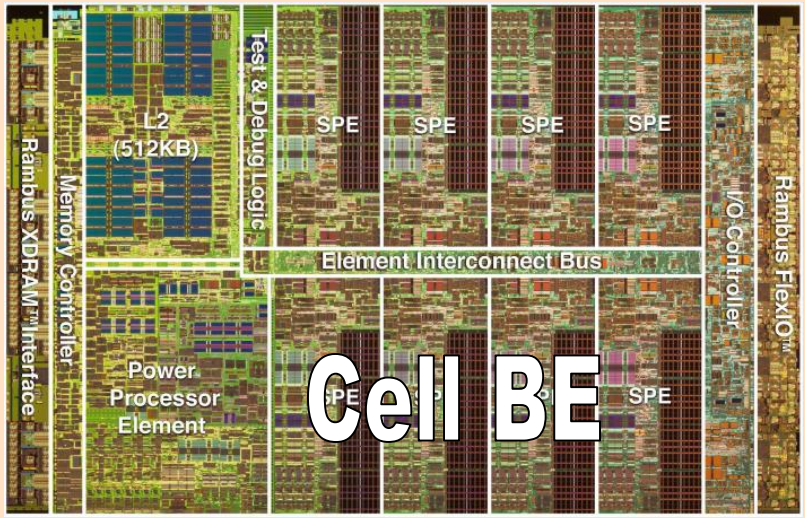


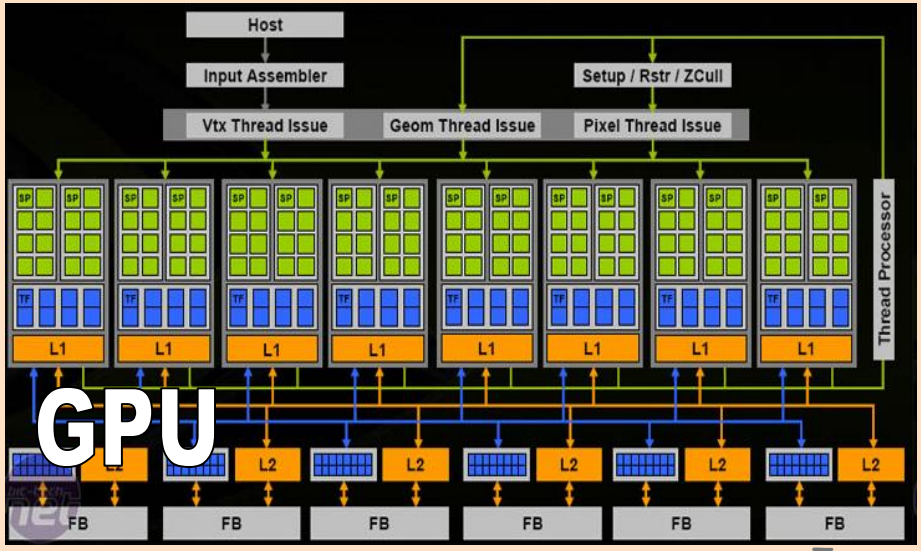
Accelerators



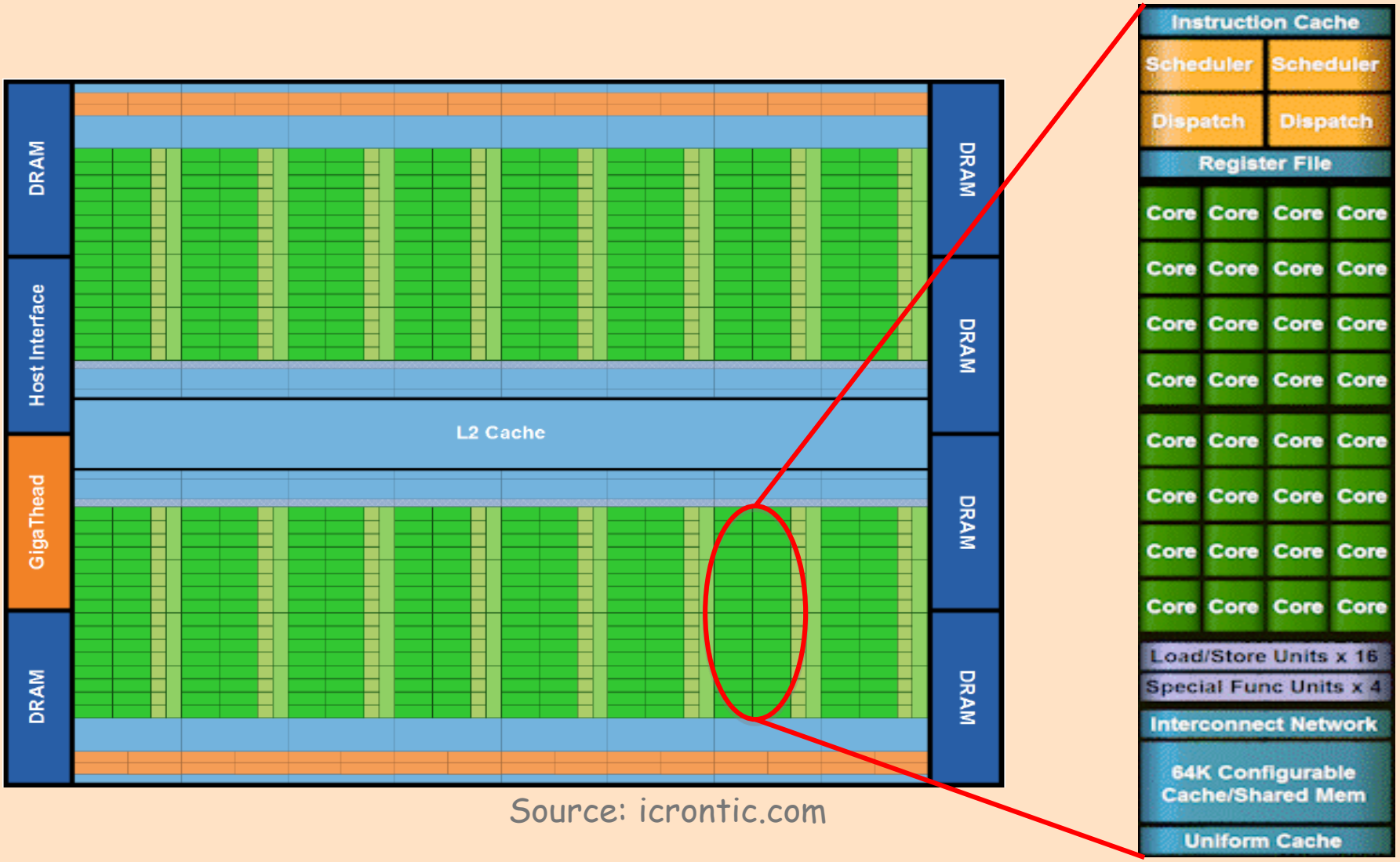
ClearSpeed CSX600







Accelerator Architecture: Fermi S2050

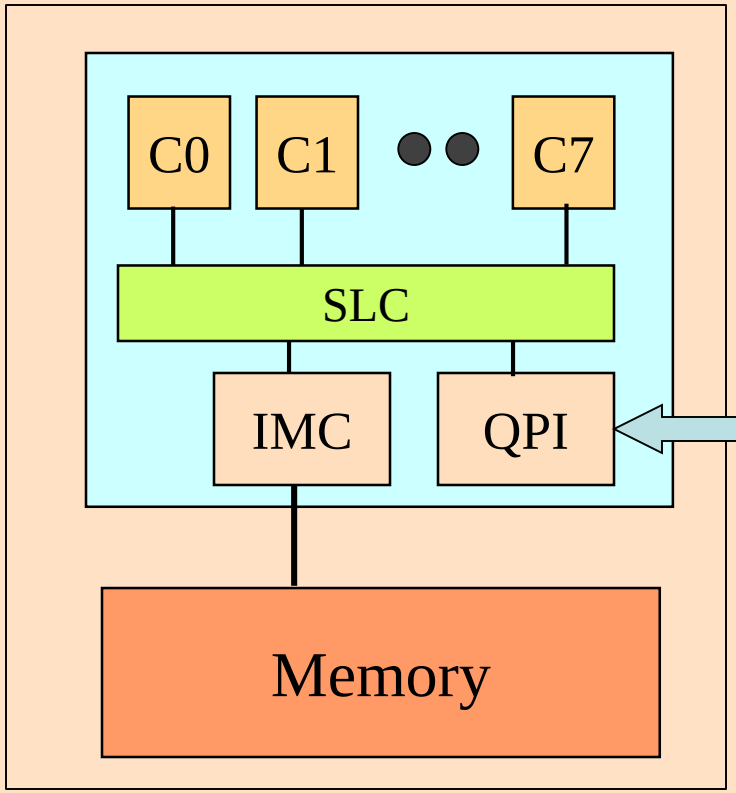


Source: icrontic.com

Comparing CPU and GPU

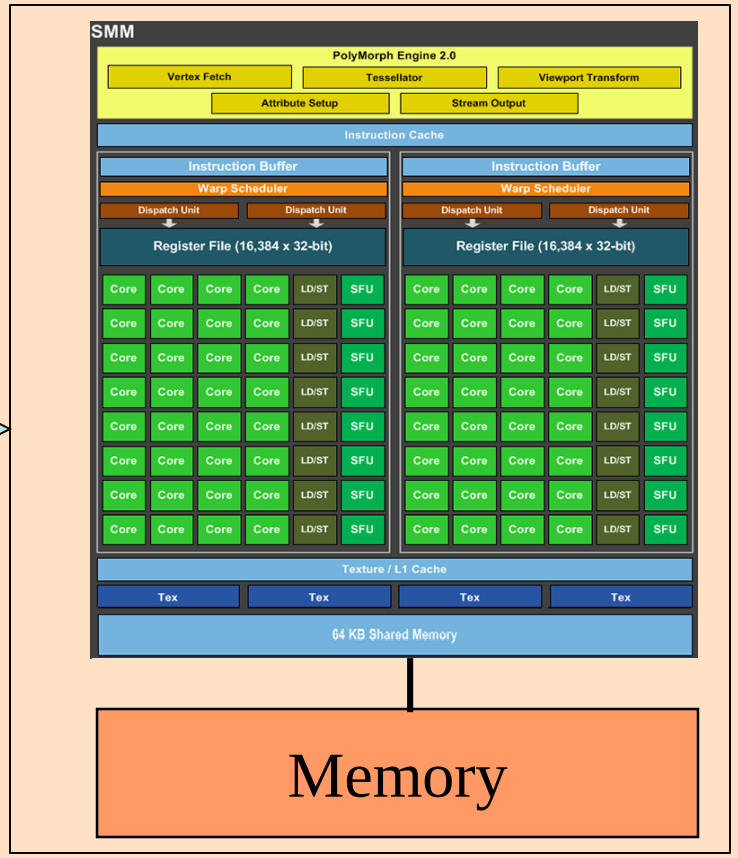
- 8 CPU cores @ 3 GHz
- 380 GFLOPS

- 2880 CUDA cores @ 1.67 GHz
- 1500 GFLOPS



CPU

PCI-e



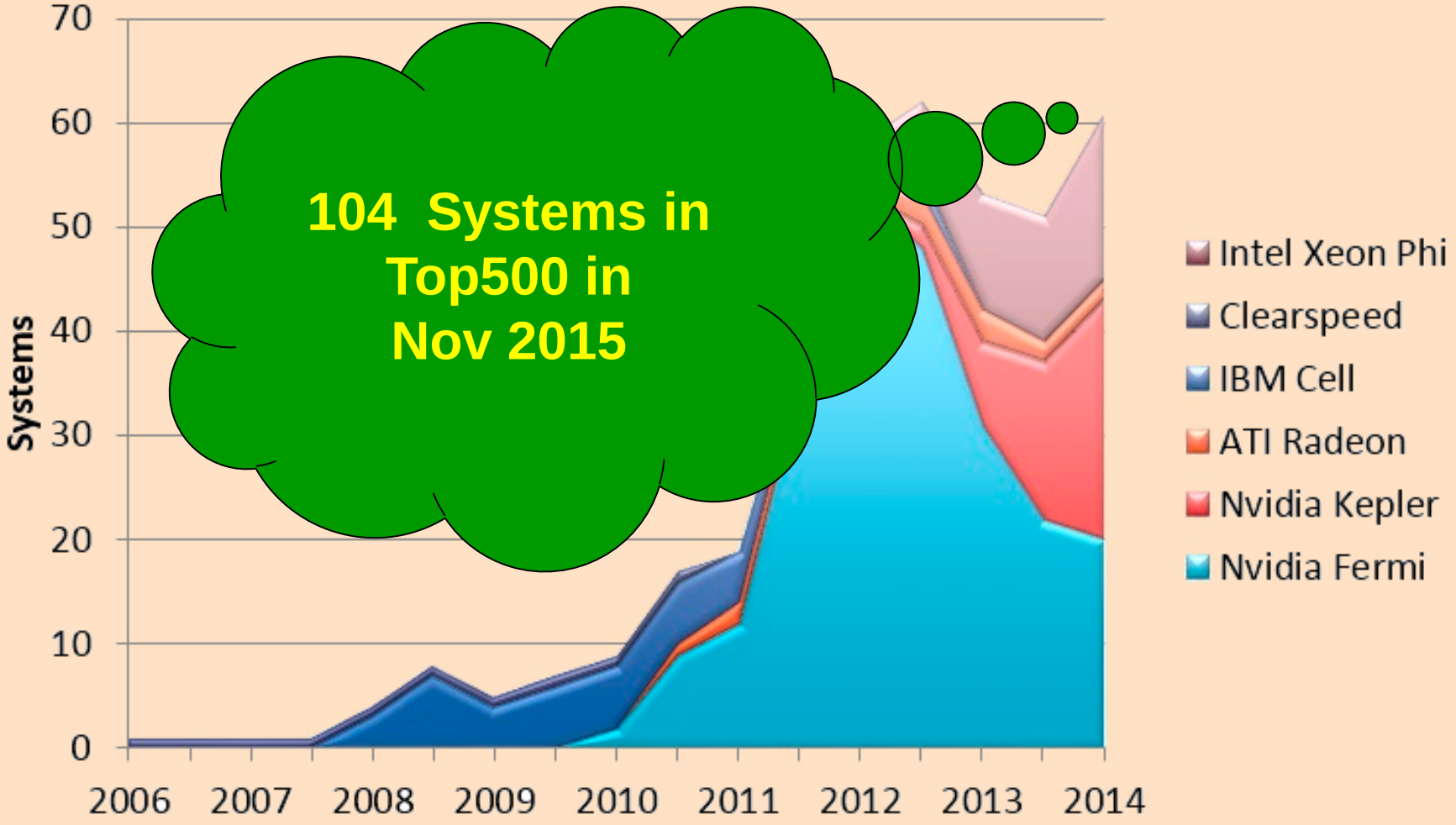
GPU

Accelerators: Hype or Reality?

Rank	Site	Manufacturer	Computer	Country	Cores	Rmax [Pflops]	Power [MW]
1	National SuperComputer Center in Tianjin	NUDT	Tianhe-2, Xeon E5 2691 and Xeon Phi 31S1	China	3,120,000	33.86	17.80
2	Oak Ridge National Labs	Cray	Titan Cray XK7, Opteron 6274 (2.2GHz) + NVIDIA Kepler K-20	USA	560,640	17.59	8.20
3	Lawrence Livermore Labs	IBM	Sequoia – BlueGene/Q	USA	1,572,864	17.17	7.89
4	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	705,024	10.51	12.66
5	DOE/SC/ANL	IBM	BlueGene/Q Power BQC 16C/1.6 GHz	USA	786,432	8.58	3.94
6	DOE/LANL/SNL	Cray	Xeon E5-2698 v3 (2.3GHz) + Aries Interconnect	USA	301,056	8.10	
7	Swiss National Computing Centre	Cray	Xeon E5-2670 (2.6GHz) + Nvidia Kepler K20x	Swiss	115,984	6.27	2.35
8	HLRS, Stuttgart	Cray	Xeon E5-2680v3 (2.5GHz) + Aries Interconnect	Germany	196,608	5.64	

Top 500 Nov 2015 List : www.top500.org

Emergence of Accelerators



Source: Nicole Hemsoth, HPC Wire, June 2014

Overview

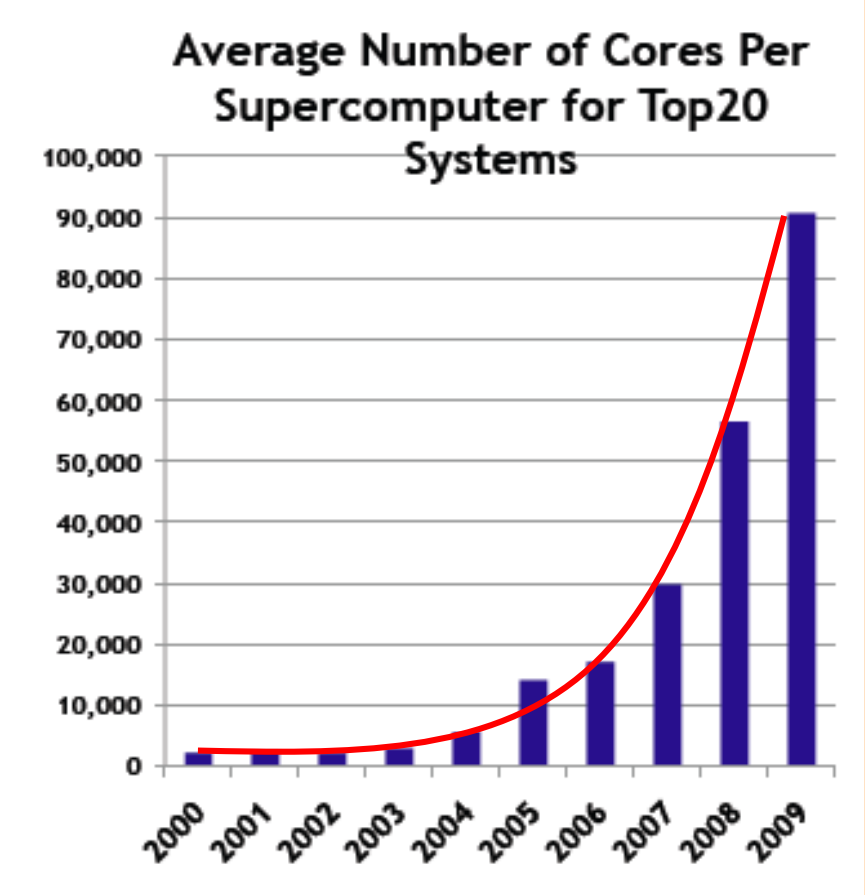
- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

Accelerator Programming: Good News

- Emergence of Programming Languages for GPUs/Accelerators
  - ➢ CUDA, OpenCL, OpenACC
  - – Efficient, but low programmability
- Growing collection of code base
  - – CUDAzone
  - – Packages supporting GPUs by Software Vendors
- Impressive performance
- What about Programmability and Portability while retaining Performance?

Handling the Multicore Challenge

- Shared and Distributed Memory Programming Languages
  - ➢ OpenMP, MPI
  - – Scaling w.r.t. no. of cores
- New Parallel Languages (partitioned global address space languages)
  - ➢ X10, UPC, Chapel, …
  - – Higher programmability - but, perf. and applicability to accelerators are issues

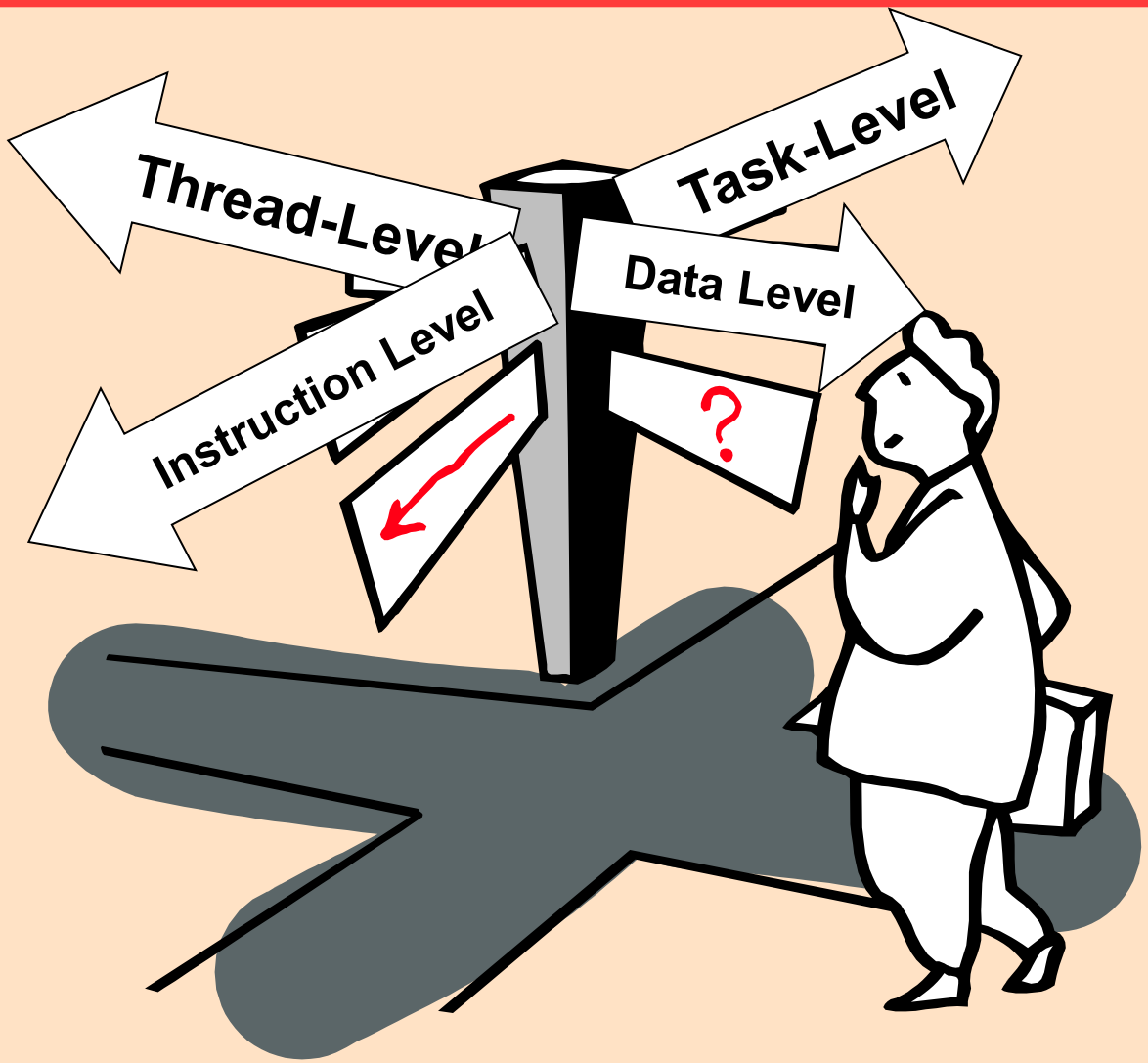


Source: Top500.org

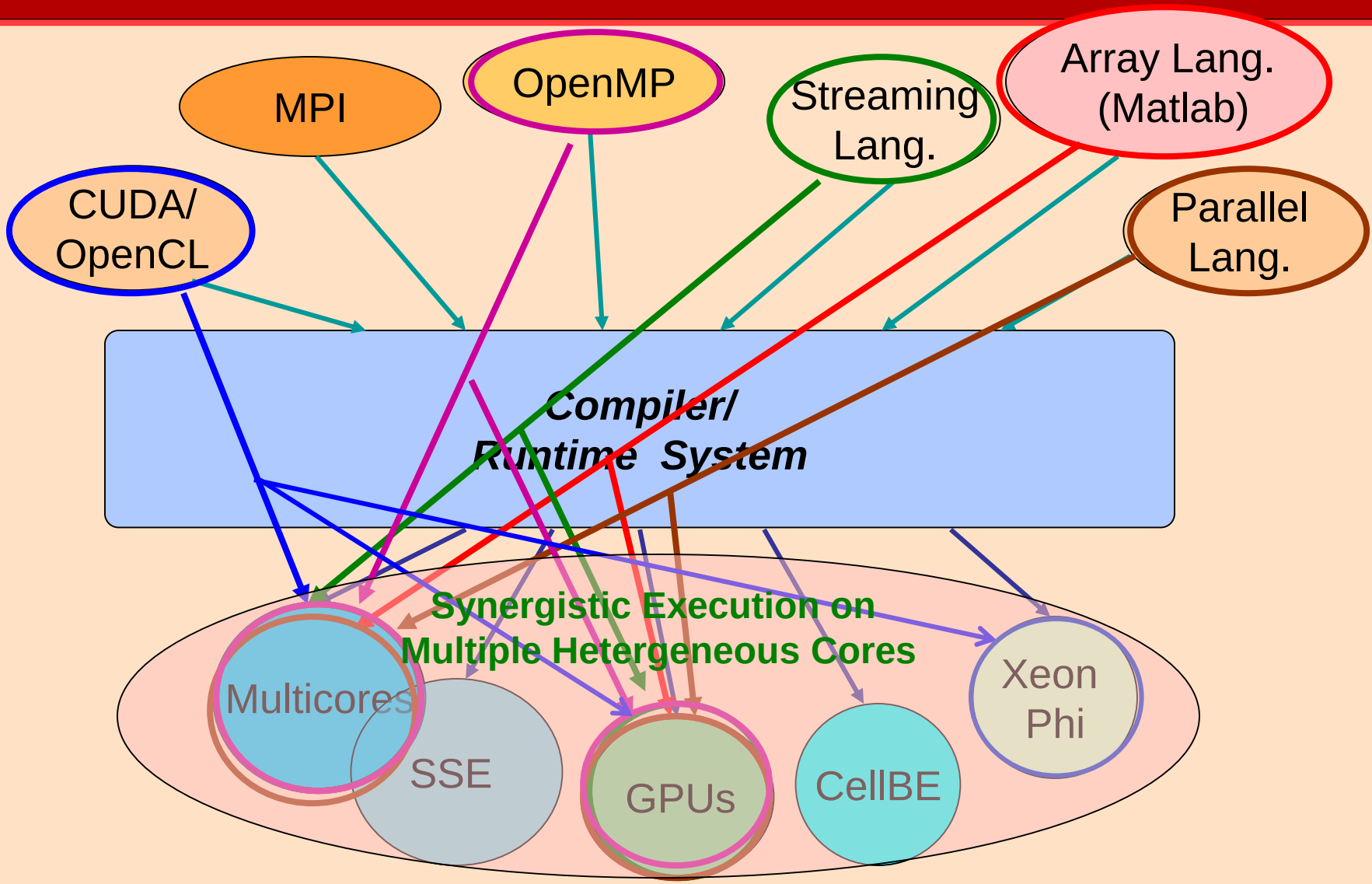
Accelerator Programming: Boon or Bane

- Challenges in programming Accelerators
  - Managing parallelism across various cores
    - Task, data, and thread-level parallelism
  - Efficient partitioning of work across different devices
  - Transfer of data between CPU and Accelerator
  - Managing CPU-Accelerator memory bandwidth efficiently
  - Efficient use of different types of memory (Device memory, Shared Memory, Constant and Texture Memory, …)
  - Synchronization across multiple cores/devices

What Parallelism(s) to Exploit?



Our Approach



Overview

- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

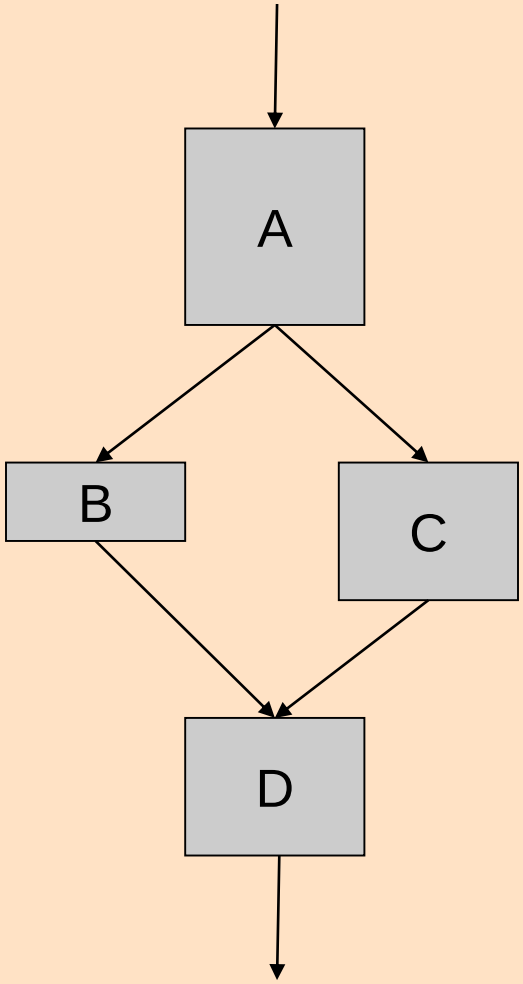
Programming Heterogeneous Systems: Our Contributions

1. Managing parallelism across Accelerator cores
   - Identifying and exposing implicit parallelism (MATLAB, StreamIT)
   - Software Pipelining in StreamIT
2. Managing data transfer between CPU and GPU
   - Compiler Scheme (MATLAB)
   - Hybrid Scheme (X10-CUDA)
3. Managing Synergistic Execution across CPU and GPU cores
   - Profile-based partitioning (StreamIT, MATLAB)
   - Runtime mechanism for OpenCL (FluidiCL)

1. Managing Parallelism across Cores

StreamIT Language

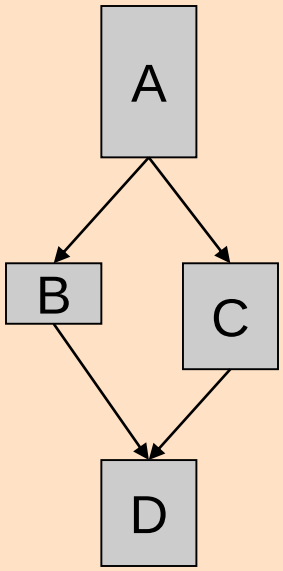
- Program is a hierarchical composition of three basic constructs
  - Pipeline
  - SplitJoin
  - Feedback Loop
- Program reprsents infinite stream of data and computation on them

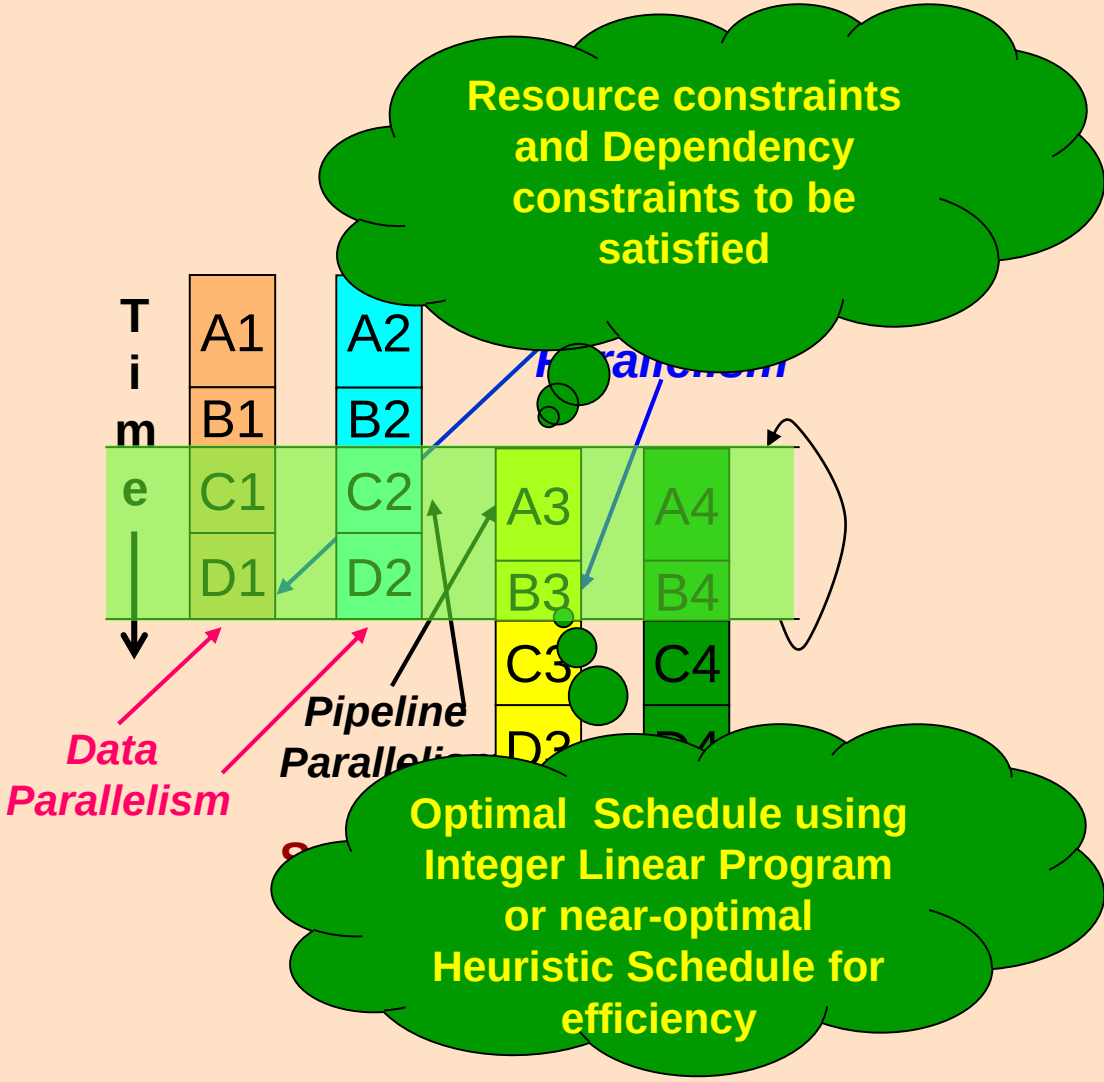


1. Managing Parallelism across Cores

- Multithreading
  - Stream of data as multiple instances of thread – data parallelism
  - Software pipelining of filters or tasks – pipelined parallelism
- Task partition between GPU and CPU cores, work scheduling and processor assignment
  - Profile-based approach for task mapping
  - Takes communication bandwidth restrictions into account
  - Task parallelism

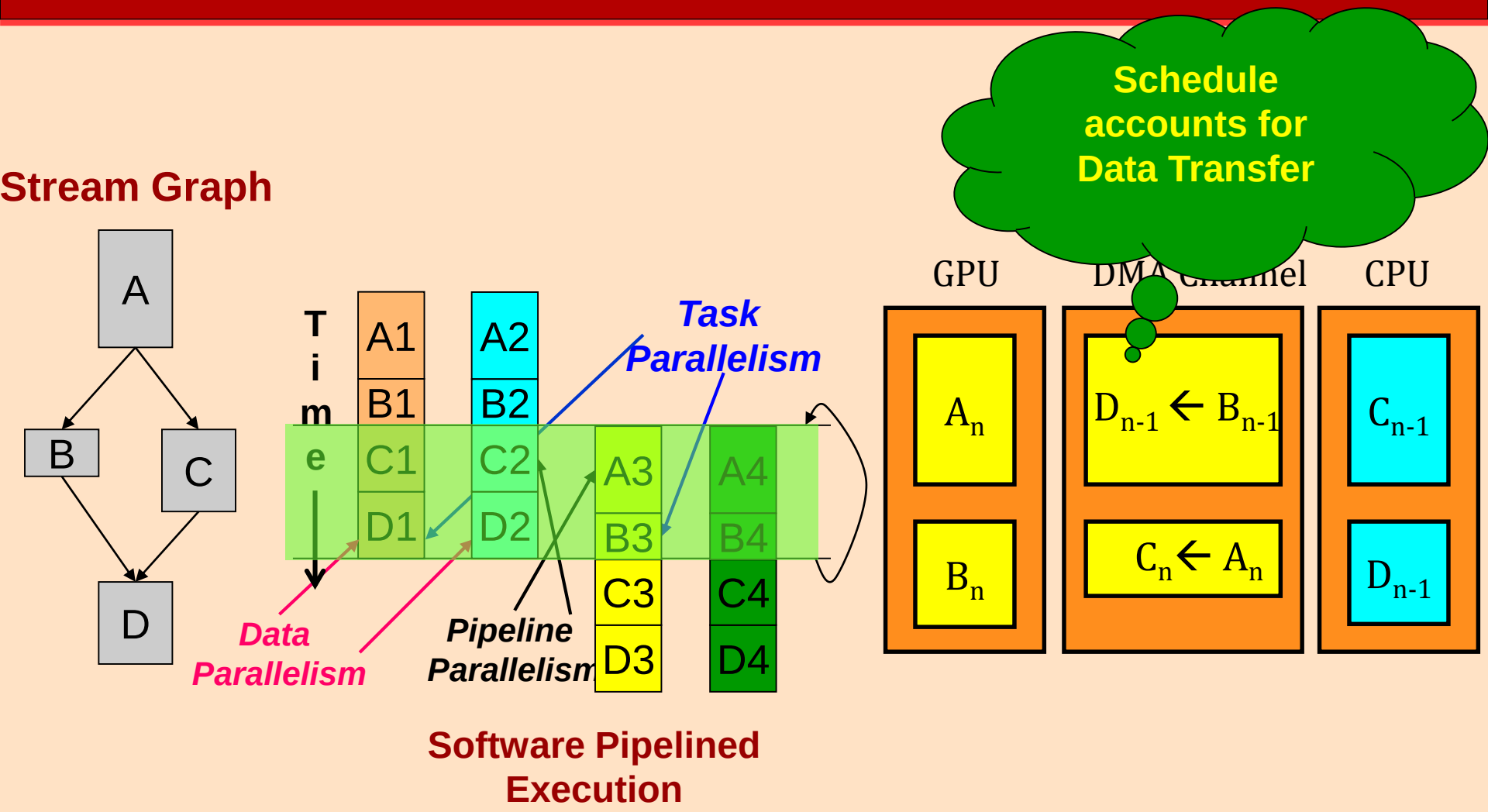
Mapping StreamIt on GPUs





[CGO 2009]

Mapping StreamIt on GPUs

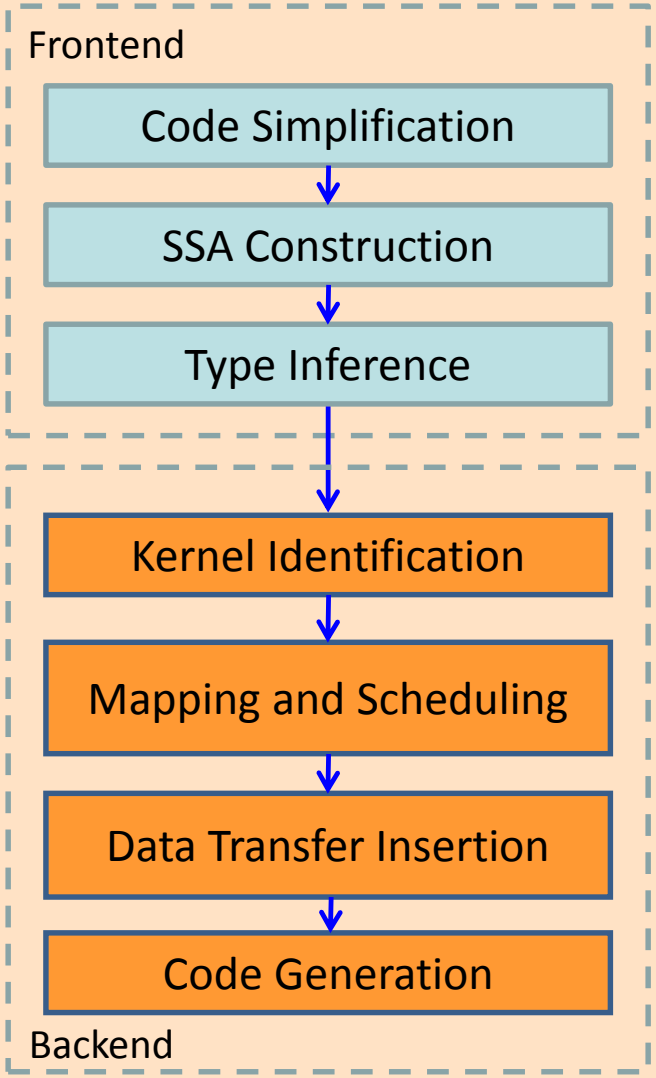


Software Pipelined Execution

MEGHA: MATLAB Execution on GPUs

MATLAB Program

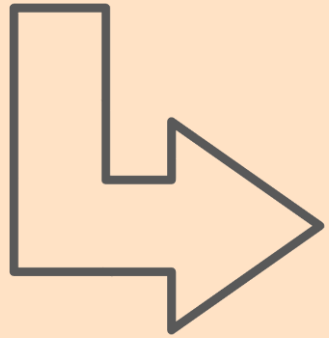
```
    array A[N][N];
    array B[N][N];
    array C[N][N]

1: tempVar0 = (B + C);
2: tempVar1 = (A + C);
3: A_1 = tempVar0 + tempVar1;
4: C_1 = A_1 * C;
```



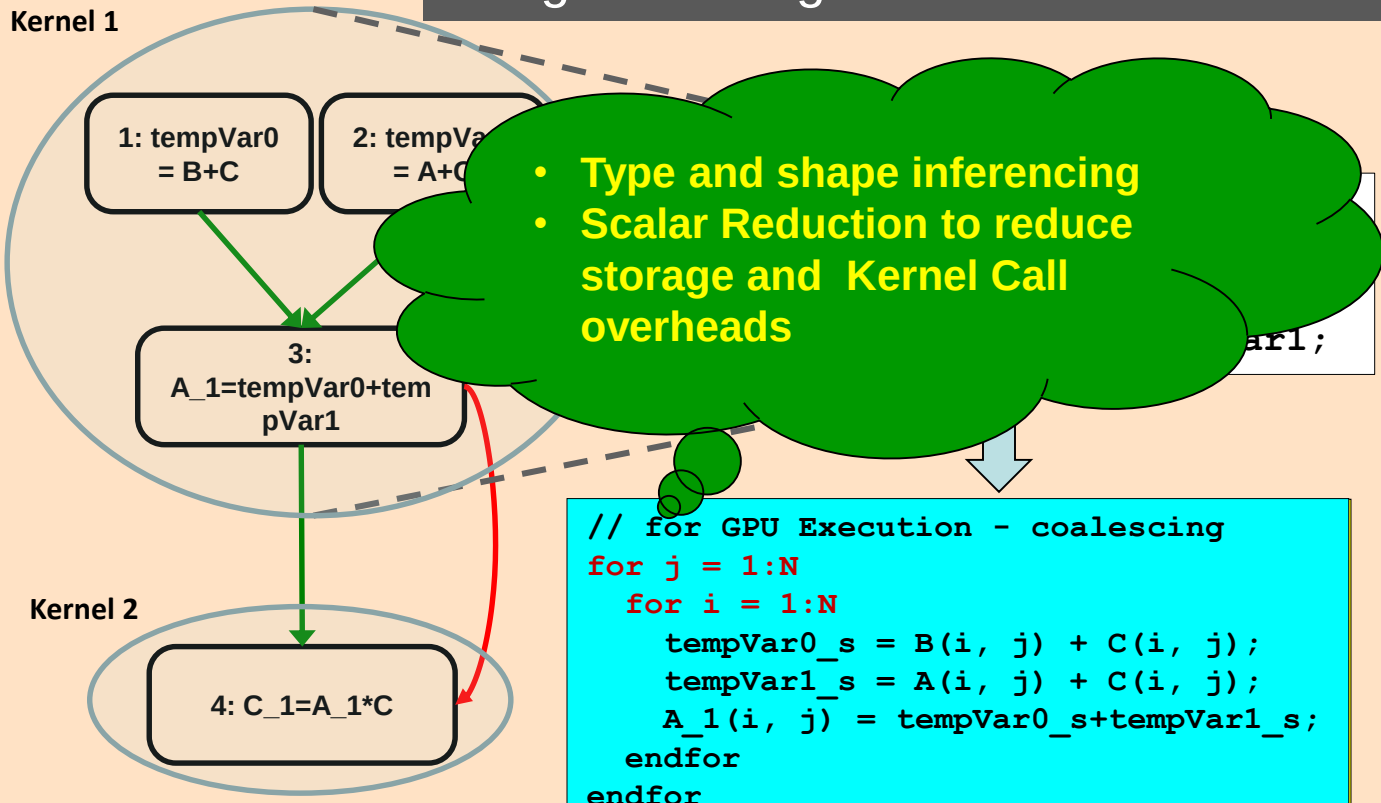
Task Partitioning and Mapping in MATLAB Execution

```
1: tempVar0 = (B + C);
2: tempVar1 = (A + C);
3: A_1 = tempVar0 + tempVar1;
4: C_1 = A_1 * C;
```

Kernel Composition : combining IR statements into **"common loops"** under memory and register constraints using Clustering Methods



Data Flow Graph Construction



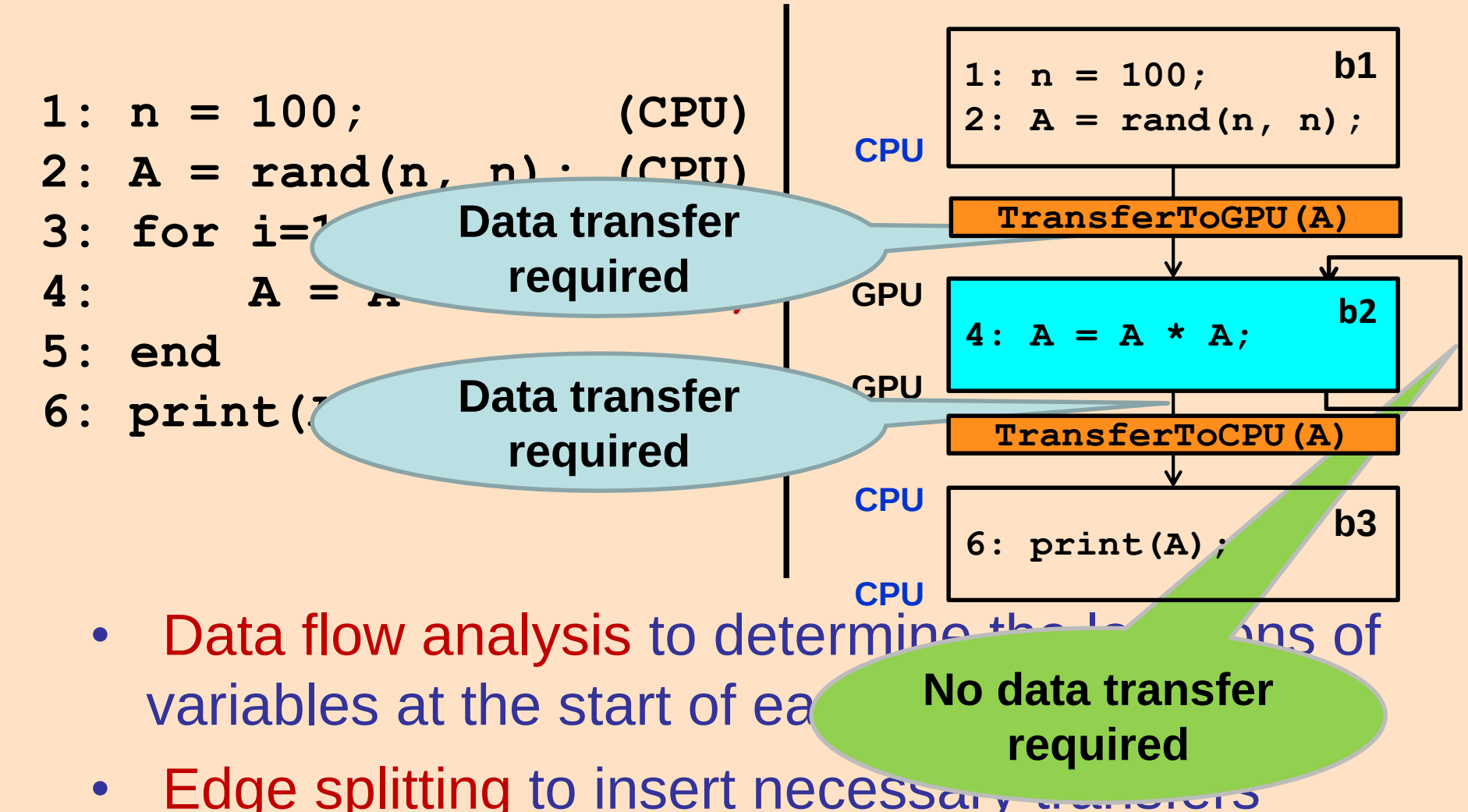

```
// for GPU Execution - coalescing
for j = 1:N
  for i = 1:N
    tempVar0_s = B(i, j) + C(i, j);
    tempVar1_s = A(i, j) + C(i, j);
    A_1(i, j) = tempVar0_s+tempVar1_s;
  endfor
endfor
```


[PLDI 2011]

Overview

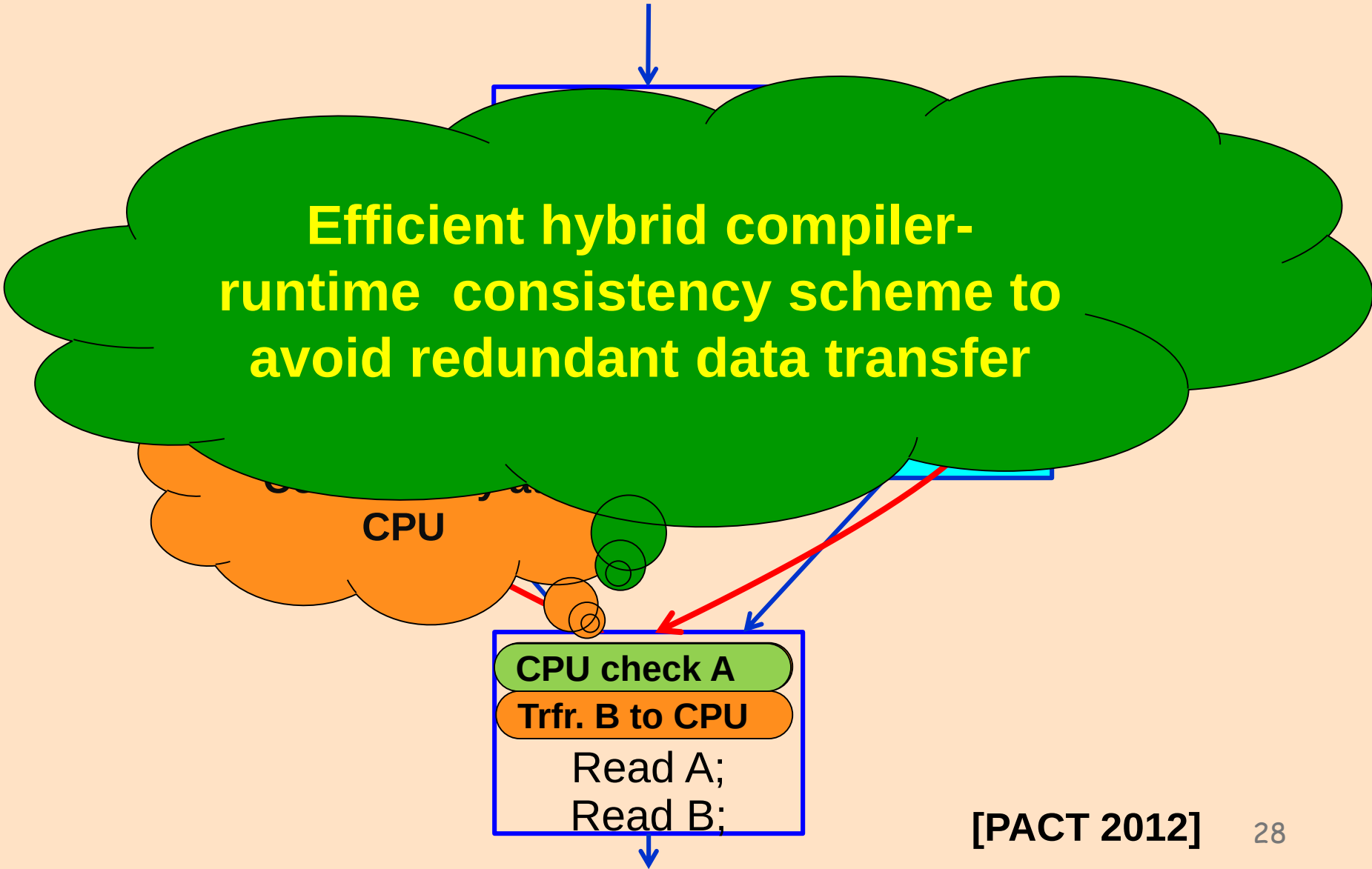
- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

2. Data Transfer Insertion



- Data flow analysis to determine the locations of variables at the start of ea
- Edge splitting to insert necessary transfers

Hybrid Compiler-Runtime Approach for Memory Consistency



[PACT 2012]

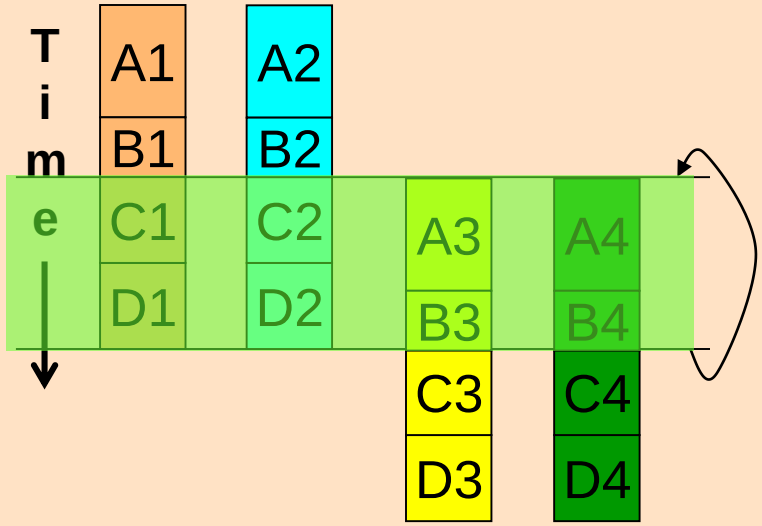
Overview

- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

3. Synergistic Execution on Heterogeneous Devices

Compile-time Approaches

- Profile based scheduling of tasks on CPU and GPU cores
- Compiler analysis of CPU- and GPU-friendly codes
- Compiler analysis for data partition and transfer



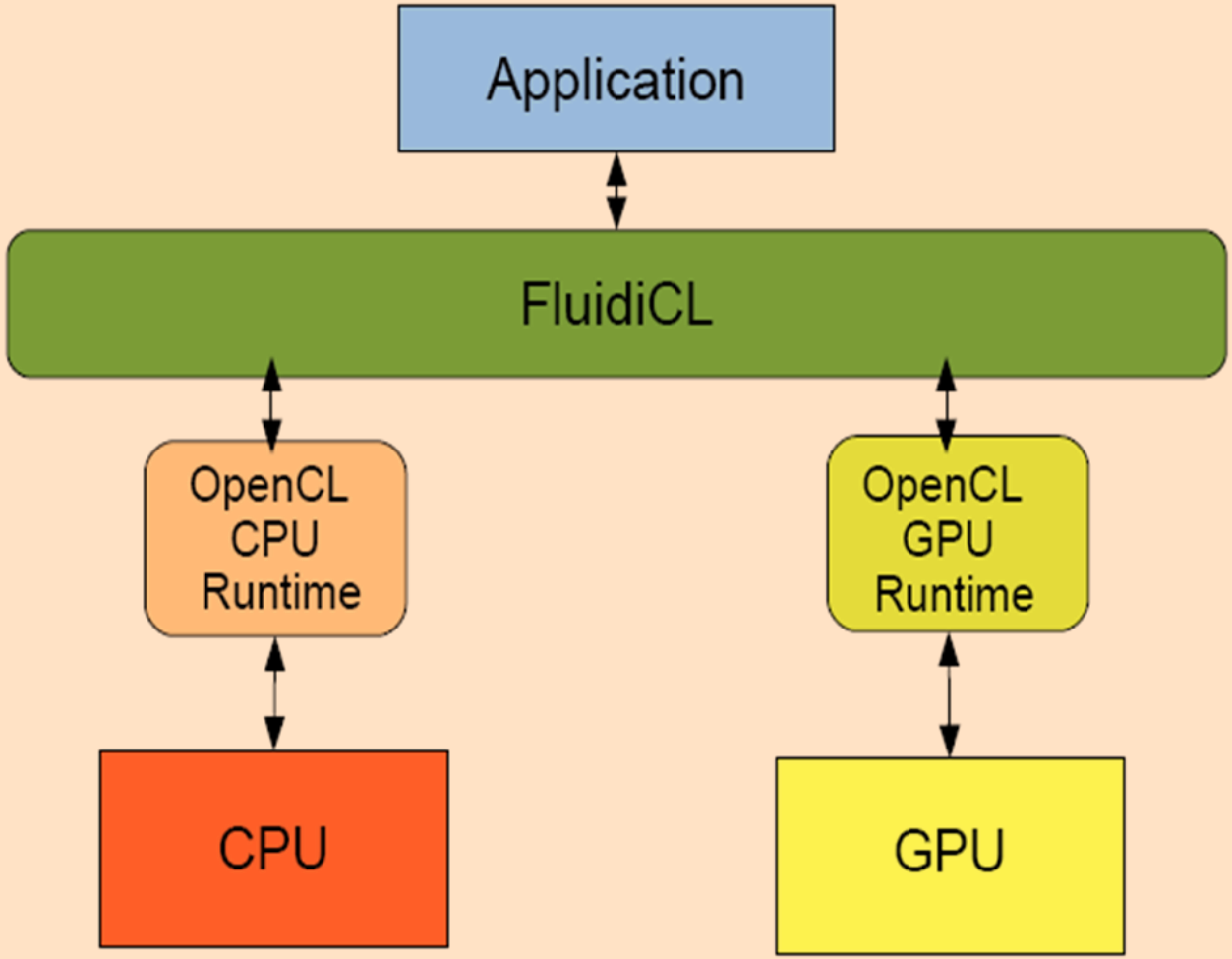
Software Pipelined Execution

3. Synergistic Execution on using Runtime Methods

- Different kernels execute better (faster) on different devices (CPU, GPU, Xeon Phi, …)
- OpenCL allows different kernels to be executed on different devices
- But require programmers to specify the device for kernel execution
- Can a **single OpenCL kernel** transparently utilize **all devices**?

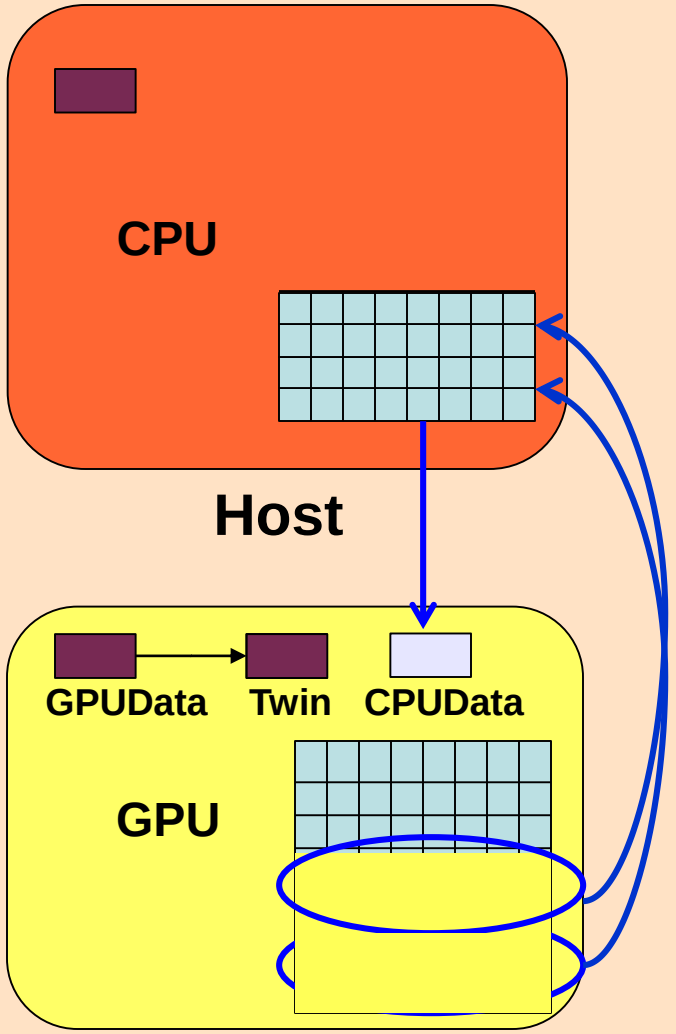
FluidiCL: A Runtime System for Cooperative and Transparent Execution of OpenCL Programs on Heterogeneous Devices

FluidiCL Runtime



Kernel Execution

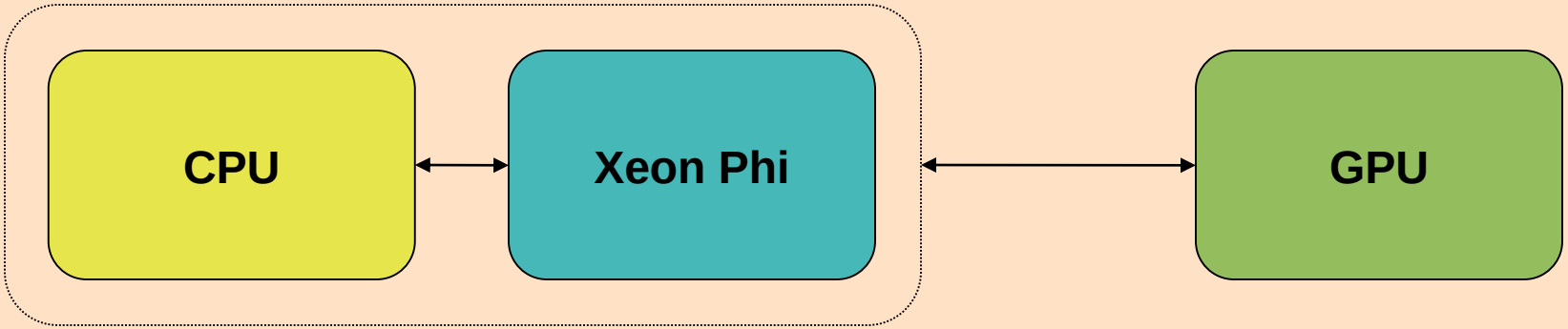
- Kernel launch is modified with a wrapper call
- Data transfers to both devices
- Two additional buffers on the GPU: one for data coming from the CPU, one to hold an original copy
- Entire grid launched on GPU
- In addition, a few work-groups, starting from end, are launched on CPU



[CGO 2014]

FluidiCL: CPU+GPU+Xeon Phi

- Consider CPU + Xeon Phi as a single device and GPU as a device
- Use FluidiCL's two-device solution between the merged device and the GPU
- Use FluidiCL's two-device solution for work distribution betwn. CPU and Xeon Phi.



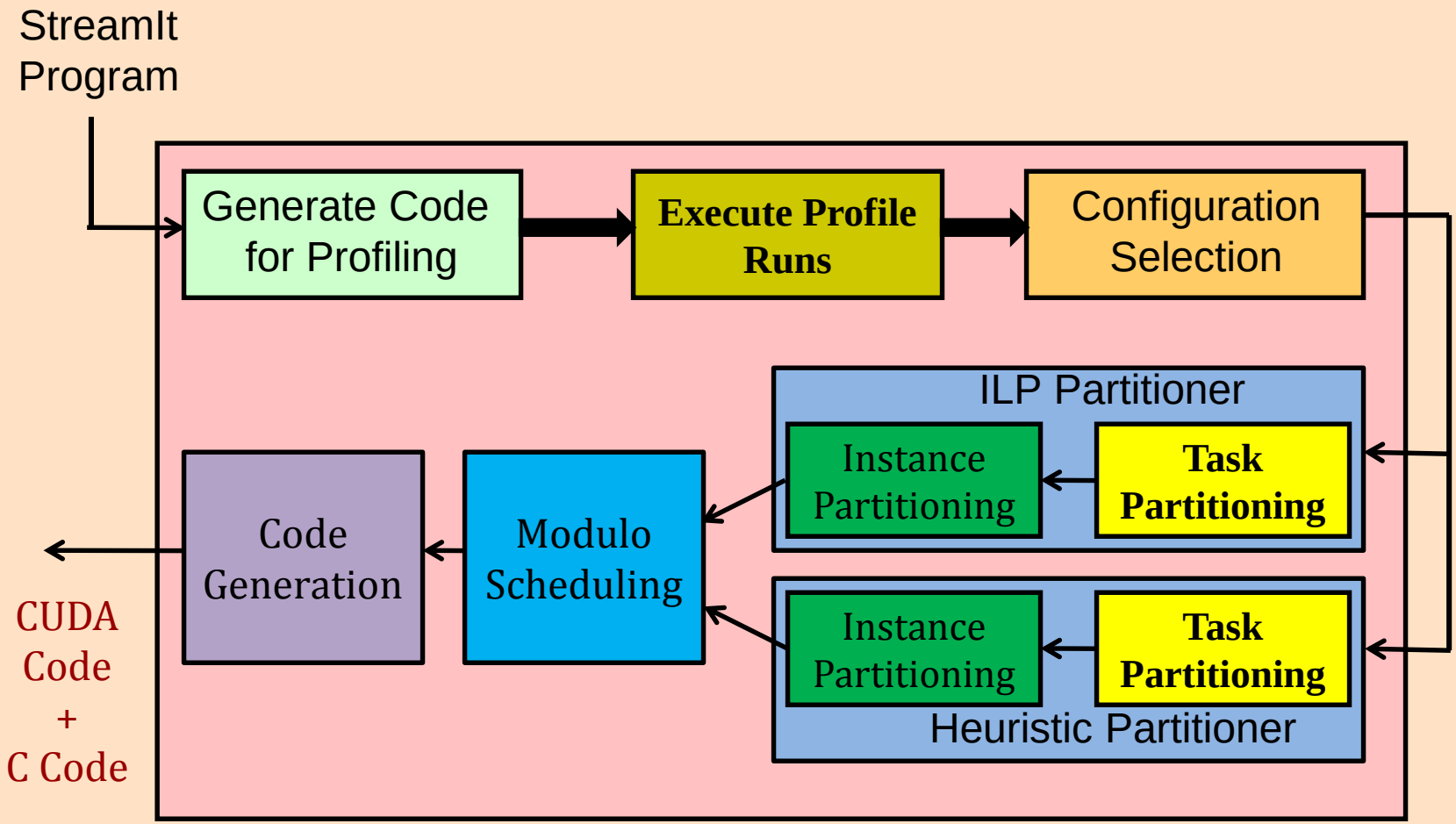
Overview

- Introduction
- Challenges in Accelerator-Based Systems
- Addressing Programming Challenges
  - Managing parallelism across Accelerator cores
  - Managing data transfer between CPU and GPU
  - Managing Synergistic Execution across CPU and GPU cores
- Implementation and Evaluation
- Conclusions

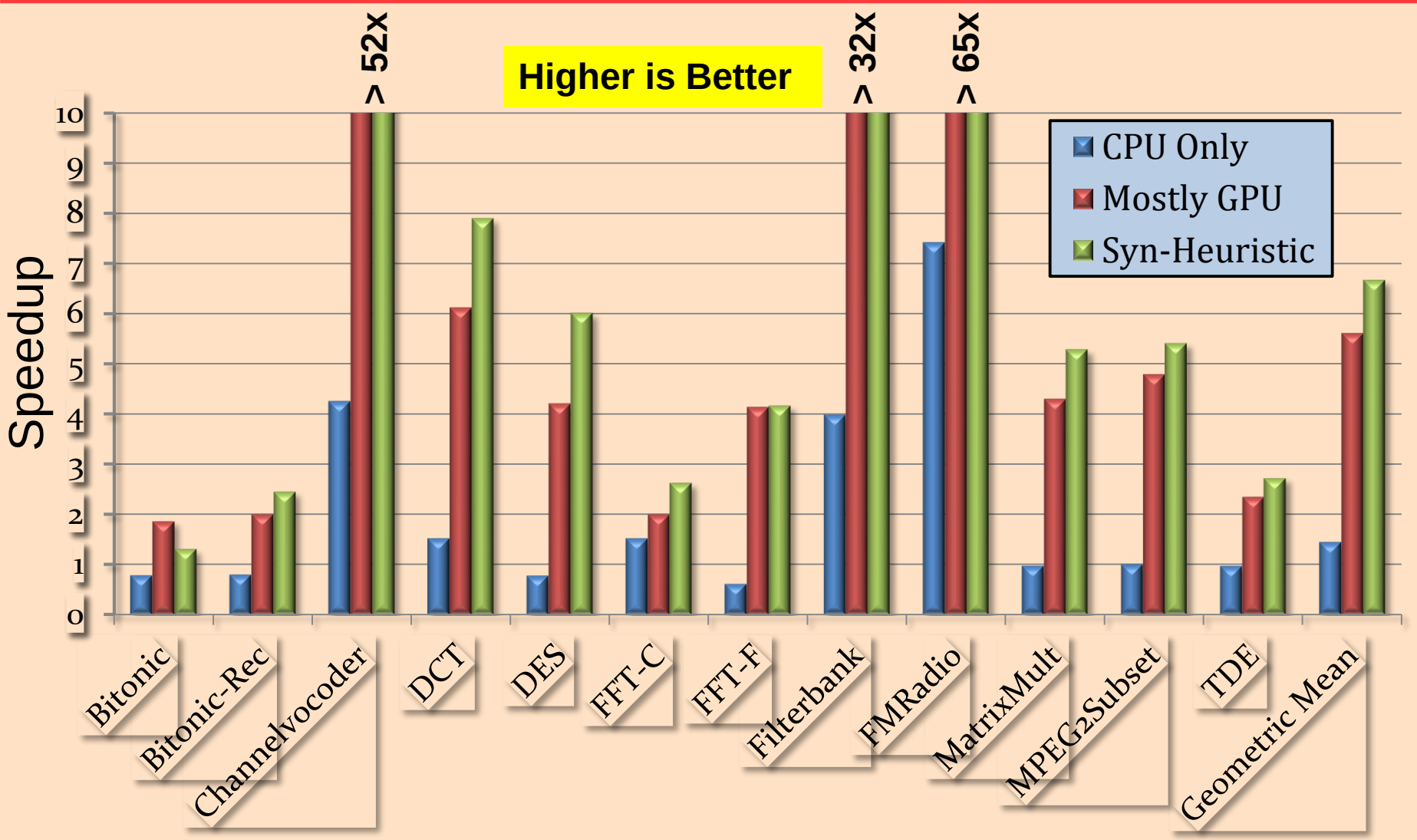
Implementation & Evaluation

- Prototype Compiler Implementation: Source-to-Source translation
  - Insertion of wrapper functions, data transfers, …
- Use of Nvidia CUDA compiler and OpenCL Compiler
- Use of CUDA and OpenCL runtimes
- Rodinia, CUDA SDK benchmarks along with benchmarks from StreamIT and MATLAB

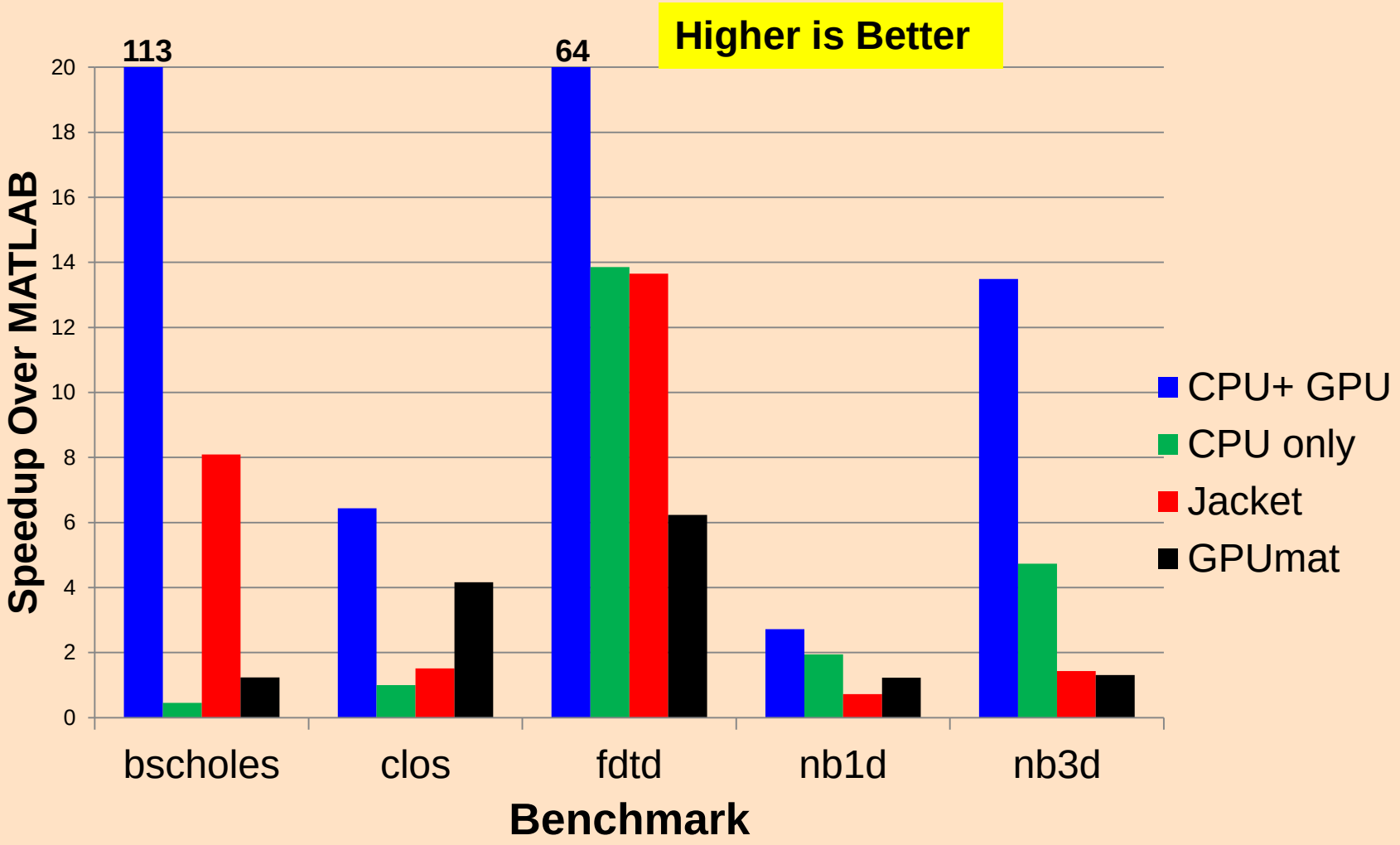
Compiler Framework for StreamIT



StreamIT: Experimental Results



MATLAB Execution : Experimental Results



Compiler Framework for X10-CUDA

X10 Code without explicit memory management

Identify variables that require to be transferred

CPU Coherence Insert checks at every read/write

GPU Coherence Insert checks before and after kernel invocation

Remove Redundant Coherence Checks

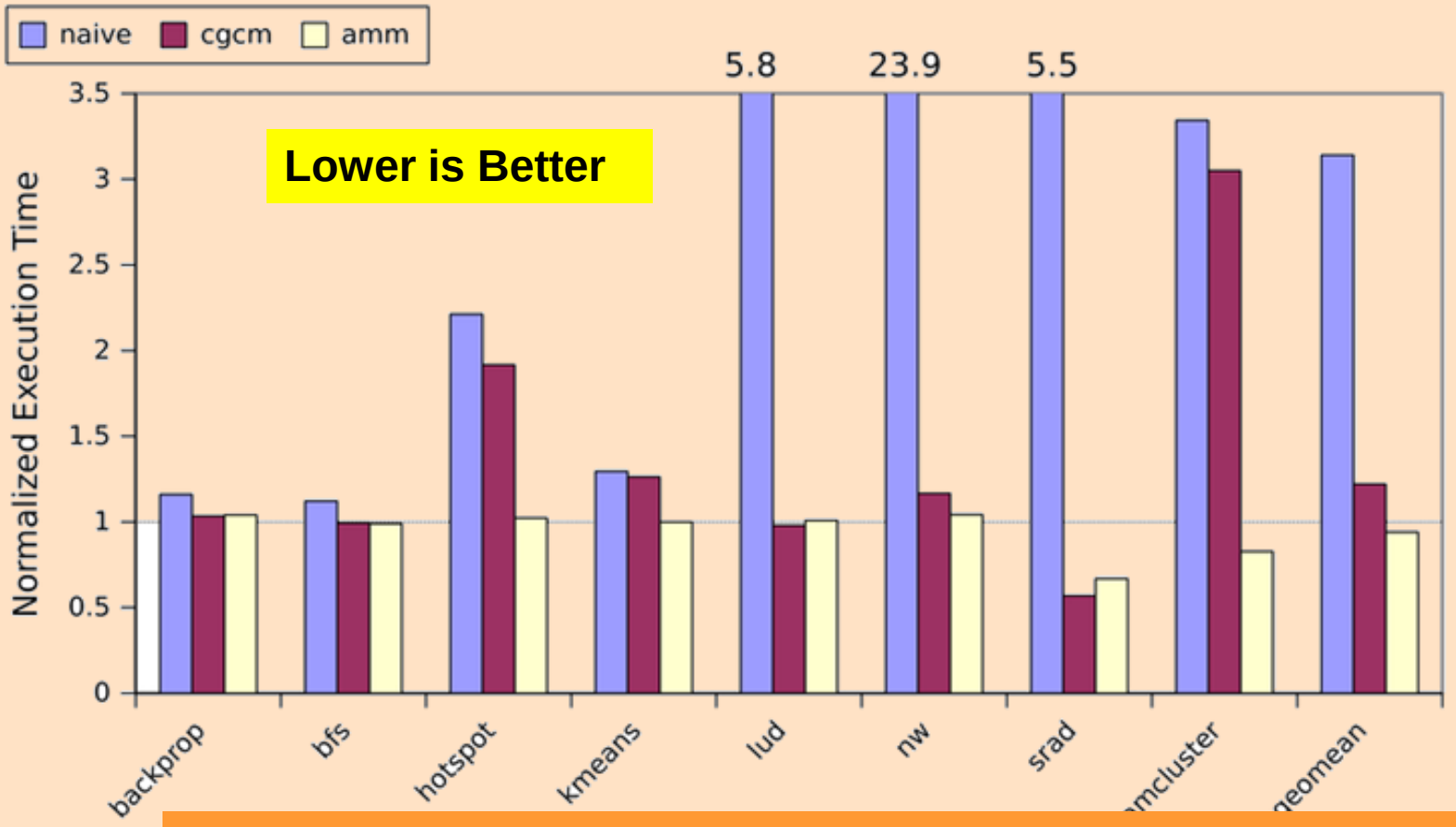
COMPILER

RUNTIME

Track Coherence Status

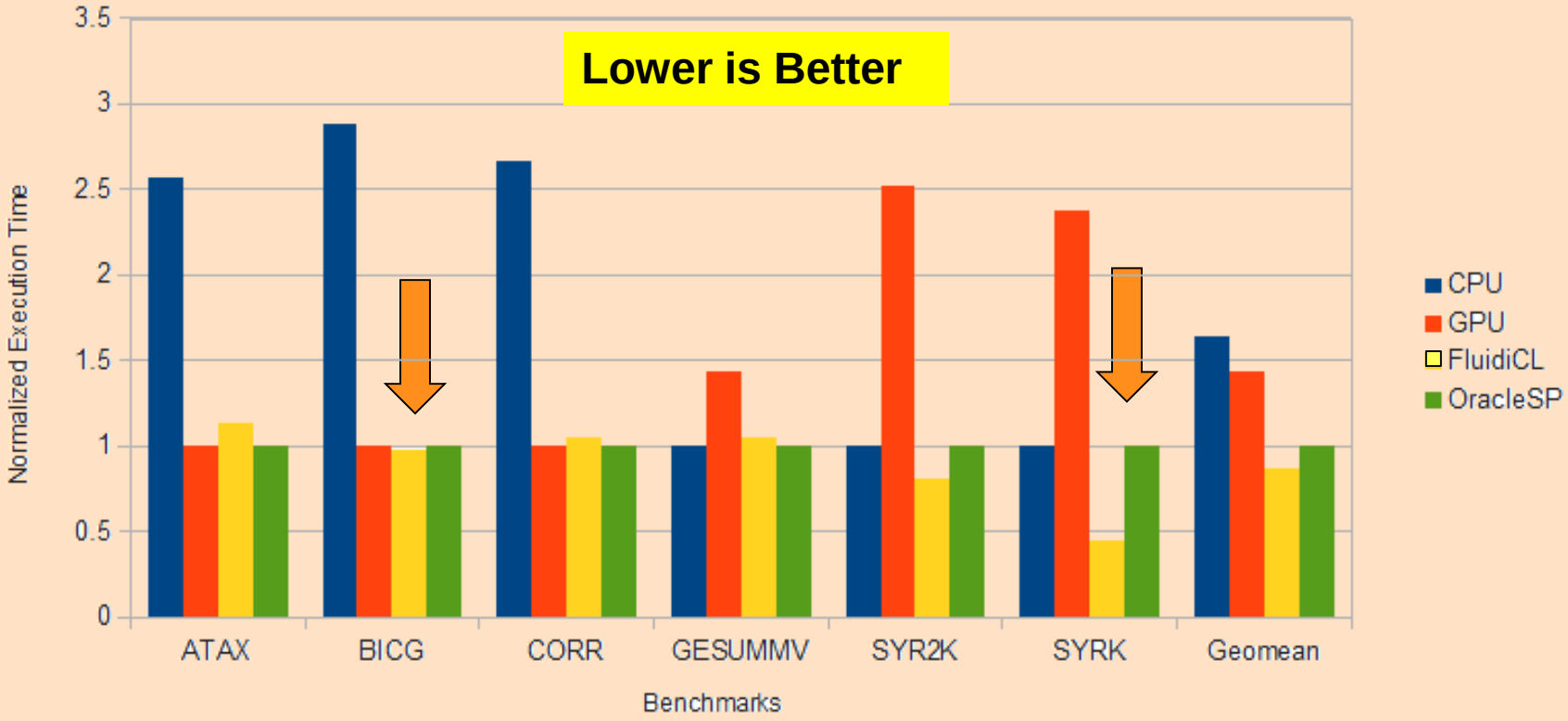
Initiate data transfers between CPU and GPU only when access to stale data is detected

Automatic Memory Management



- 3.33X faster than naïve
- Comparable to hand-tuned (explicit) or better
- 1.29X faster than CGCM

Transparent Kernel Execution



- FluidiCL execution times comparable to the best of the devices
- Better than best-device performance in two applns.

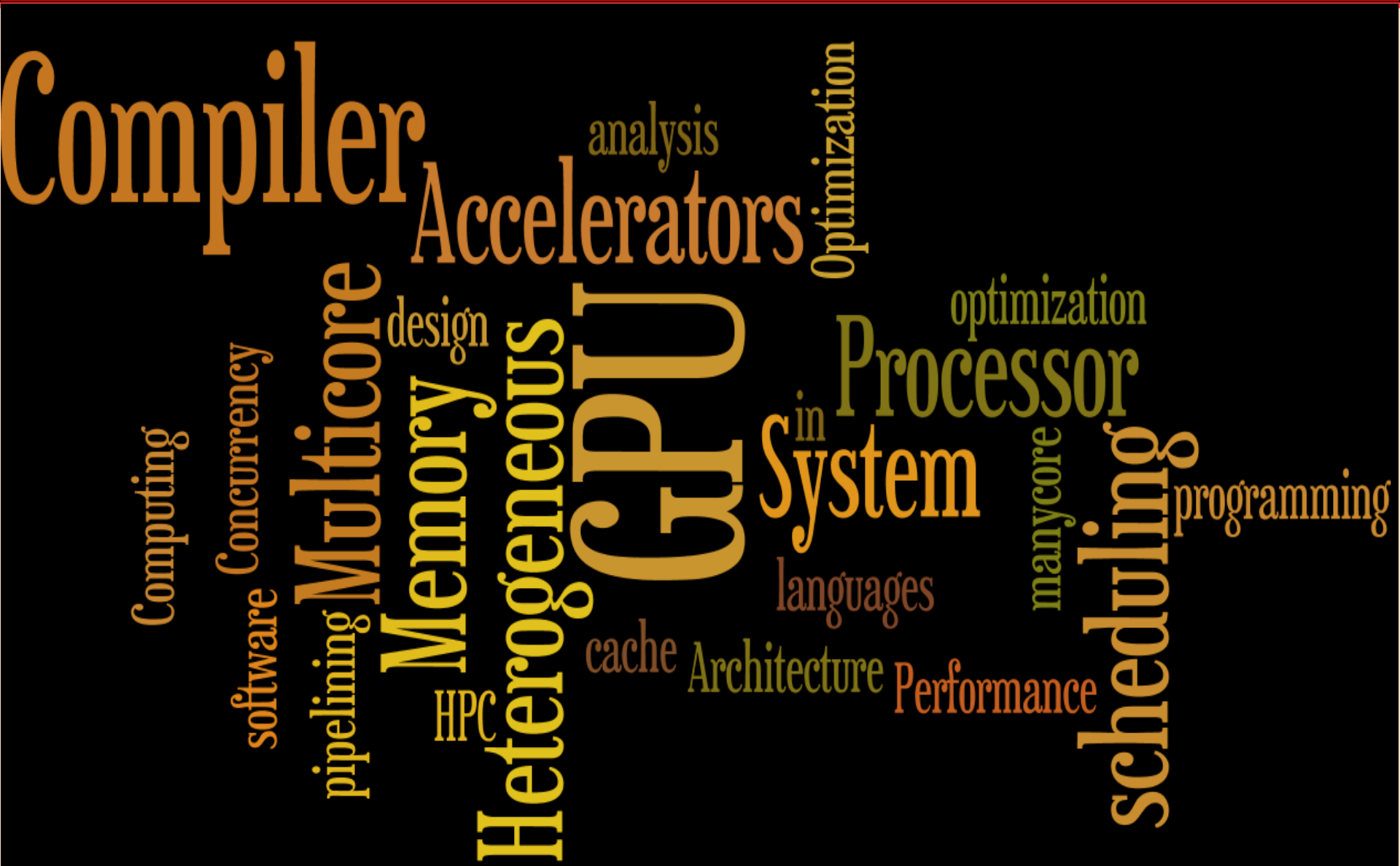
Summary

- Another exciting decade is awaiting for Accelerators
- Need innovations in
  - Compilers
  - Programming Languages & Models
  - Runtime Systems
- Key Challenges are: Programmability, Portability and Performance

Acknowledgements

- My Students
  - Abhishek Udupa, Ashwin Prasad, Jayvant Anantpur, Prasanna Pandit, Radhika Sarma, Raghu Prabhakar, Sreepathi Pai
  - Kaushik Rajan, R. Manikantan, Nagendra Gulur
- Funding Agencies
  - AMD, IBM, Intel, Microsoft, Nvidia

Computer System Research @HPC Lab



Thank You !!

Dec. 2015