Mapping a Data-Flow Programming Model onto Heterogeneous Platforms

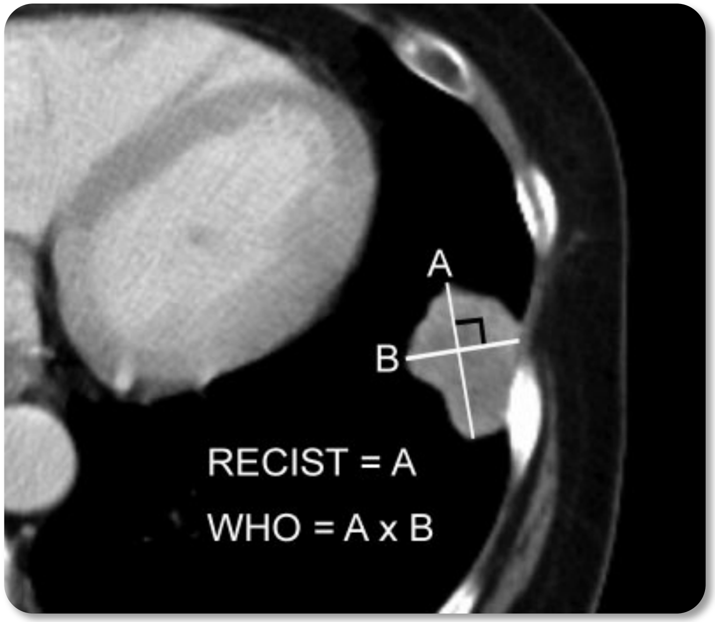
Alina Sbîrlea[α], Yi Zou[β], Zoran Budimlić[α],
Jason Cong[β], Vivek Sarkar[α]

Rice University[α]   UCLA[β]

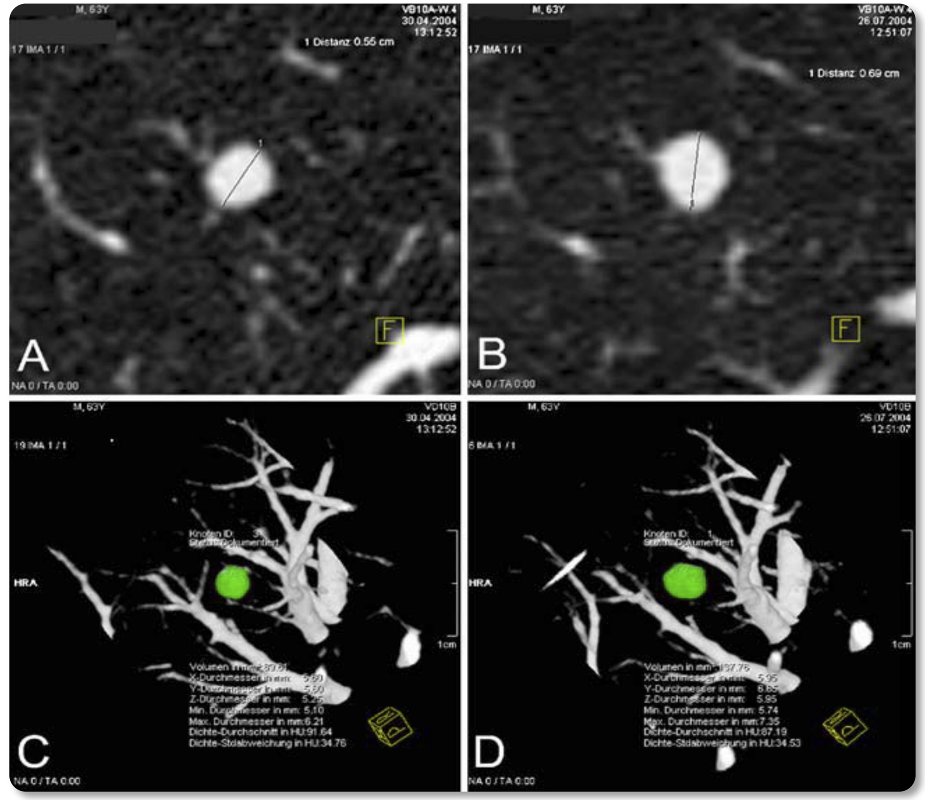
Objective

- High level data flow model
  - Domain experts
- Hybrid architectures
- Results:
  - Increased performance
  - Low energy consumption
  - High programmability

Motivation: Medical imaging applications



Suzuki C, et al.
Source: NSF CDSC project, Application thrust



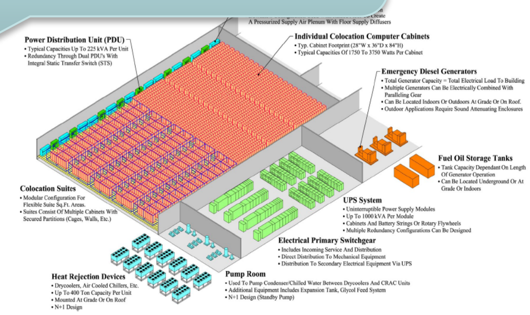
A 63-year old patient with solitary pulmonary metastasis from renal cell cancer. Manual unidimensional measurements documented a growth of 25%, consistent with stable disease (**a,b)**. However, automated volumetry documented a volume growth of 53.8%, consistent with progressive disease.

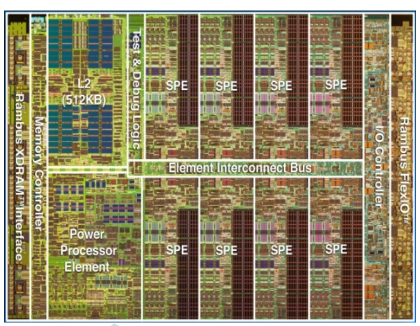
Marten K et al. Eur Radiol (2006) 16:781-90. Source: NSF CDSC project, Application thrust

Heterogeneous resources

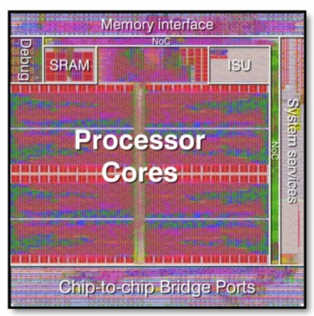
Multicore CPU

Clusters and data centers



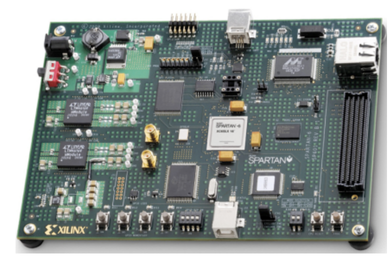


Specialized microprocessors



Embedded data parallel coprocessor

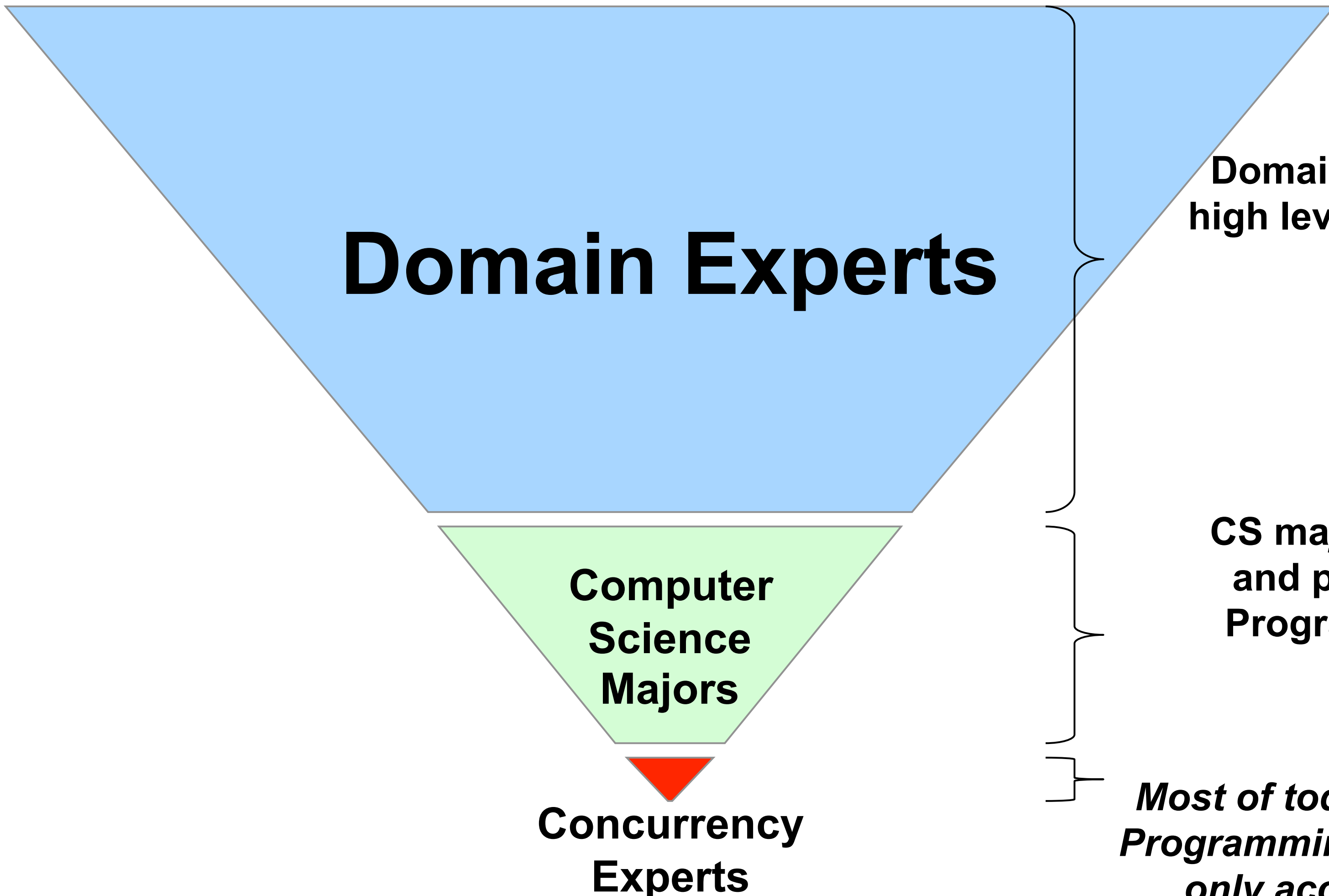
Graphical processing units (GPU)



Field-programmable gate arrays (FPGAs)

Figure Sources: Habanero team, cpu-world.com, NVIDIA, Xilinx

Programming models



Domain Experts need high level Programming Models

CS majors need simple and portable Parallel Programming Models

Most of today's Parallel Programming Models are only accessible to concurrency experts

Slide credit: Habanero team

Proposed solution

- Concurrent Collection (CnC) programming model
  - Clear separation between application description and implementation
  - Fits domain expert needs

- CnC-HC: Software flow CnC => Habanero-C(HC)
- Cross-device work-stealing in Habanero-C
  - Task affinity with heterogeneous components
- Data driven runtime in CnC-HC
- Real application – medical imaging domain

Outline

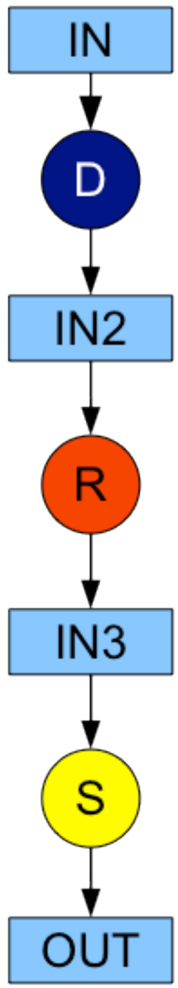
- Concurrent Collections programming model
- Programming model and runtime extensions
- Target platform
- Experimental results
- Conclusions and future work

CnC Building Blocks

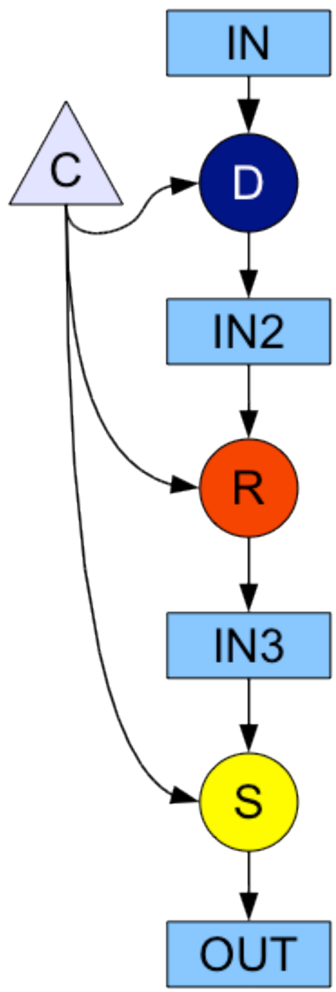
- Steps
  - Computational units
  - Functional with respects to their inputs
- Data Items
  - Means of communication between steps
  - Dynamic single assignment
- Control Items
  - Used to create (prescribe) instances of a computation step

CnC – Building a graph

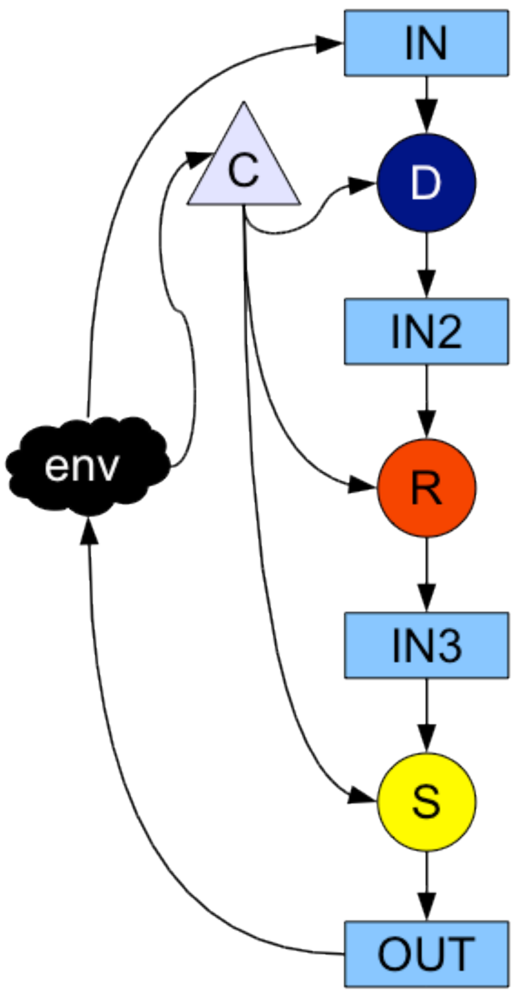
CnC – Building a graph



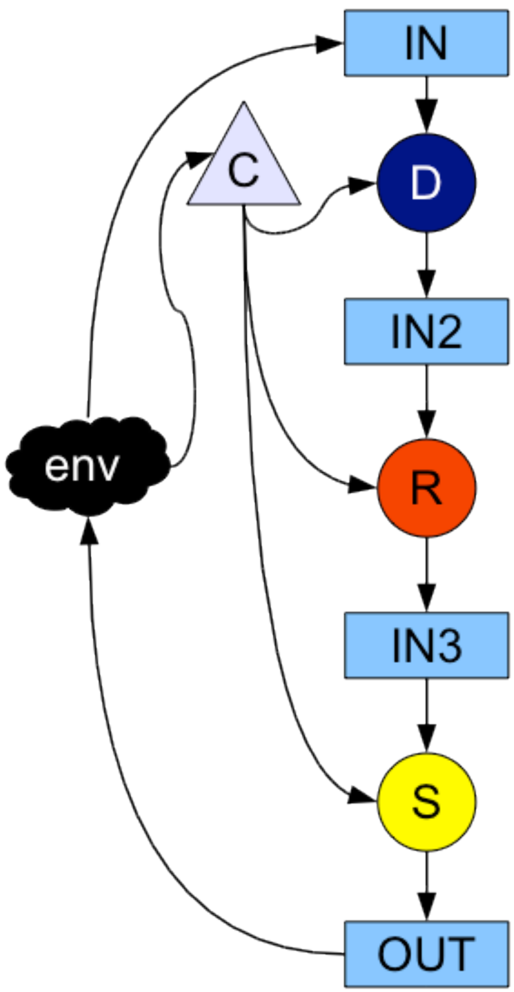
CnC – Building a graph



CnC – Building a graph

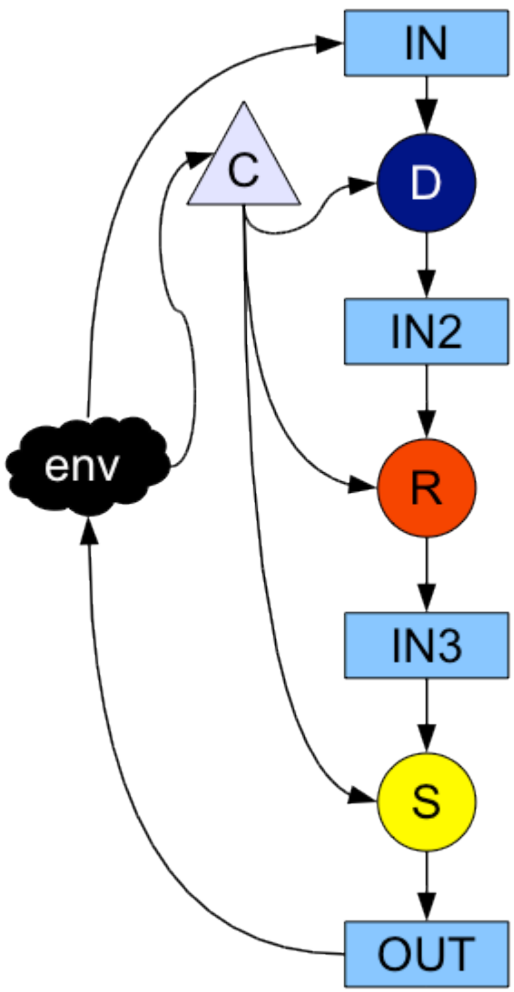


CnC – Building a graph



- Textual graph representation:
  - < C > :: (D);
  - < C > :: (R);
  - < C > :: (S);
  - [IN] → (D) → [IN2];
  - [IN2] → (R) → [IN3];
  - [IN3] → (S) → [OUT];
  - env → [IN], < C >;
  - [OUT] → env;

CnC- extending the model (1/2)

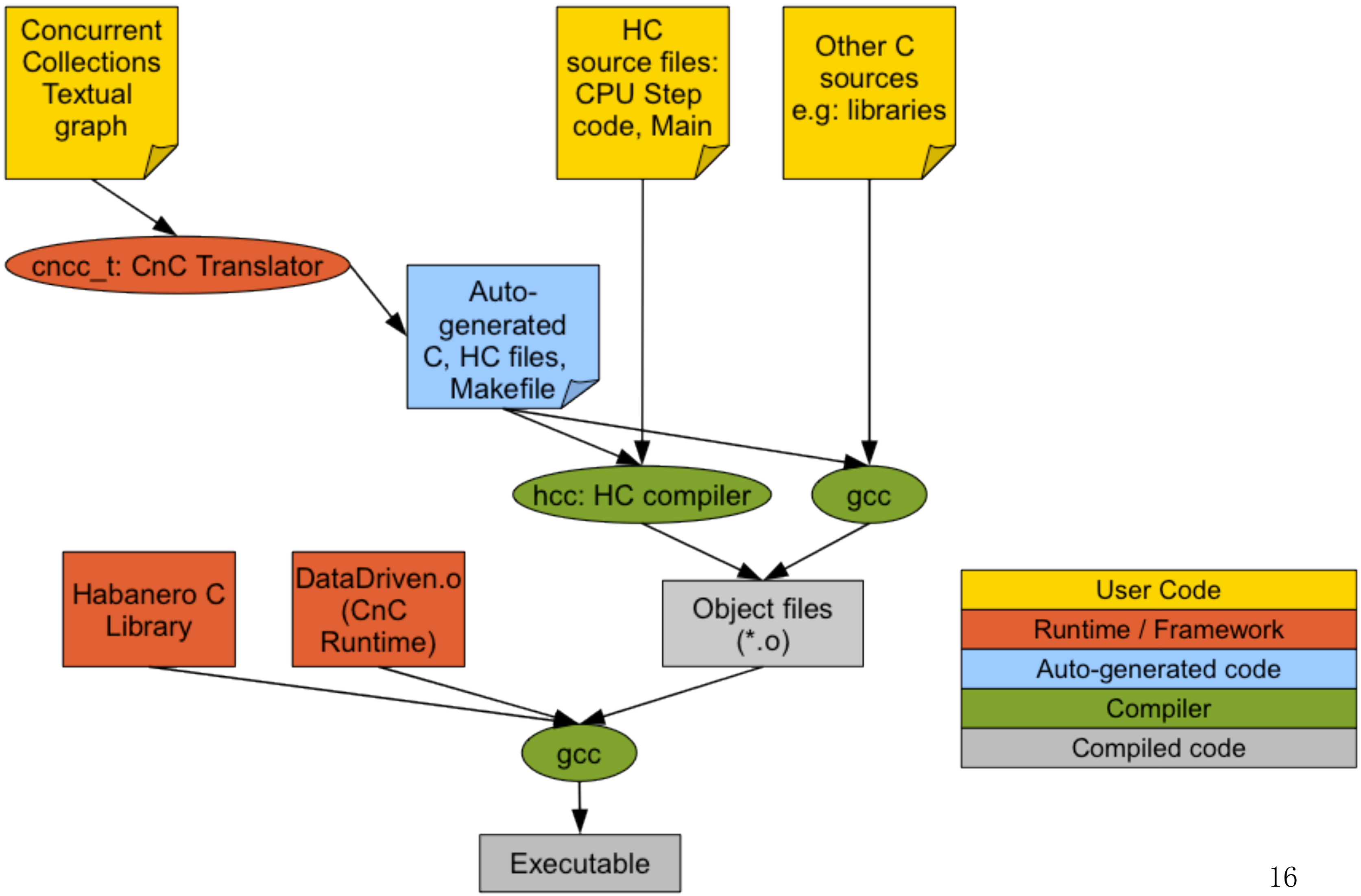


- Textual graph representation with tag functions and ranges:
  - < C > :: (D);
  - < C > :: (R);
  - < C > :: (S);
  - [IN : k-1] → (D : k) → [IN2 : k+1];
  - [IN2 : 2*k] → (R : k) → [IN3 : k/2];
  - [IN3 : k] → (S : k) → [OUT : IN3[k]];
  - env → [IN : { 0 .. 9 }], < C : { 0 .. 9 } >;
  - [OUT : 1] → env;

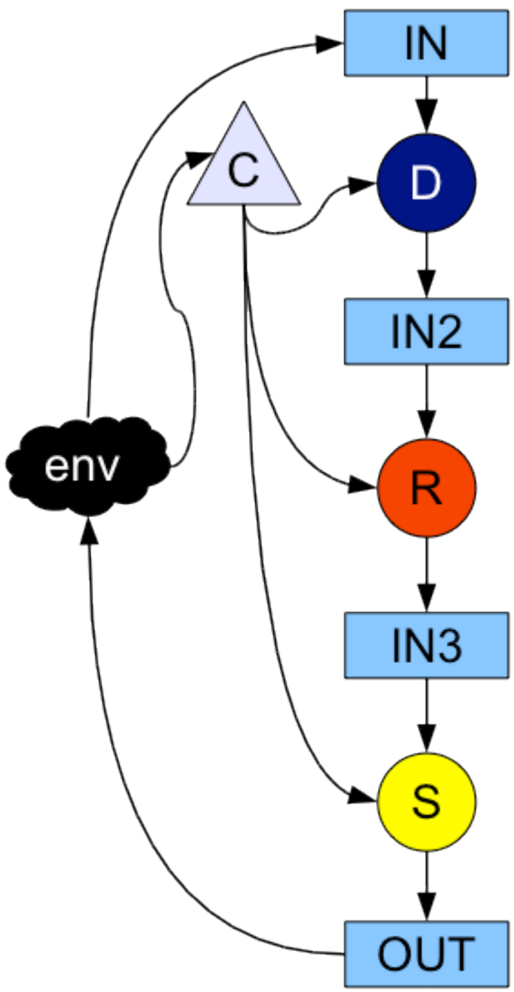
Why tag functions?

- Tag function = a mapping from what uniquely identifies a step to its inputs/outputs
- Increase programmability
- Facilitate code generation
- Enable a more efficient data-driven runtime use for step scheduling
- Many other research opportunities
  - Dependency graph analyzable
  - Graph correctness
  - Memory management

Translating CnC specifications



CnC- extending the model (2/2)



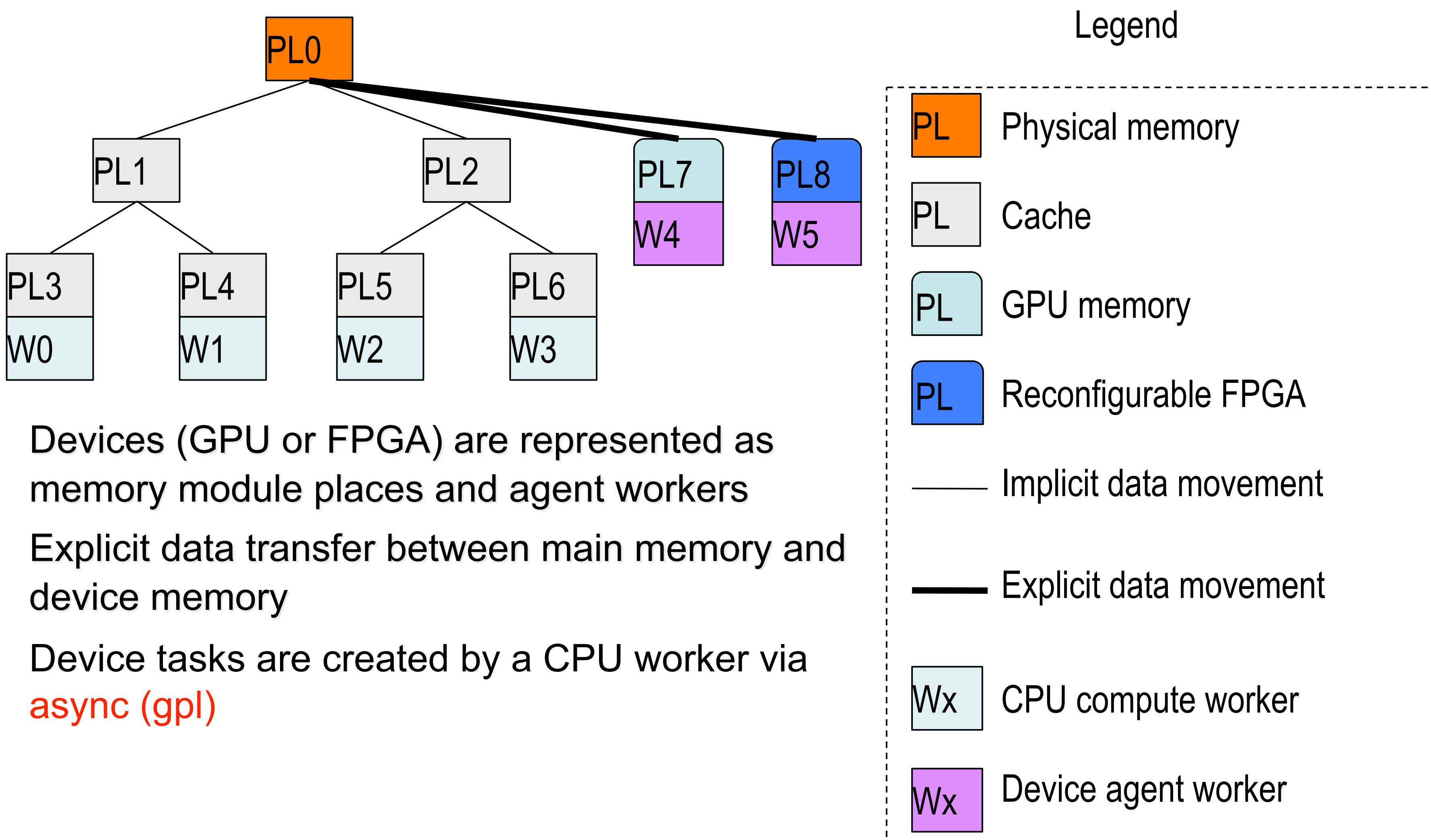
- Textual graph representation with tag functions and affinity annotations:
  - < C > :: (D @CPU=20,GPU=10);
  - < C > :: (R @GPU=5, FPGA=10);
  - < C > :: (S @GPU=12);
  - [IN : k-1] → (D : k) → [IN2 : k+1];
  - [IN2 : 2*k] → (R : k) → [IN3 : k/2];
  - [IN3 : k] → (S : k) → [OUT : IN3[k]];
  - env → [IN : { 0 .. 9 }], < C : { 0 .. 9 } >;
  - [OUT : 1] → env;

Habanero-C (HC) language

- Async and Finish constructs
  - Work-stealing
- Hierarchical place trees (HPTs)
  - Task Locality
  - XML

```xml
<HPT version="0.1">
    <place num="1" type="cpu" size="16G">
        <core num="2"/>
    </place>
     <place num="1" type="fpga" size="16G">
     </place>
     <place num="1" type="nvgpu" size="4G">
     </place>
</HPT>
```

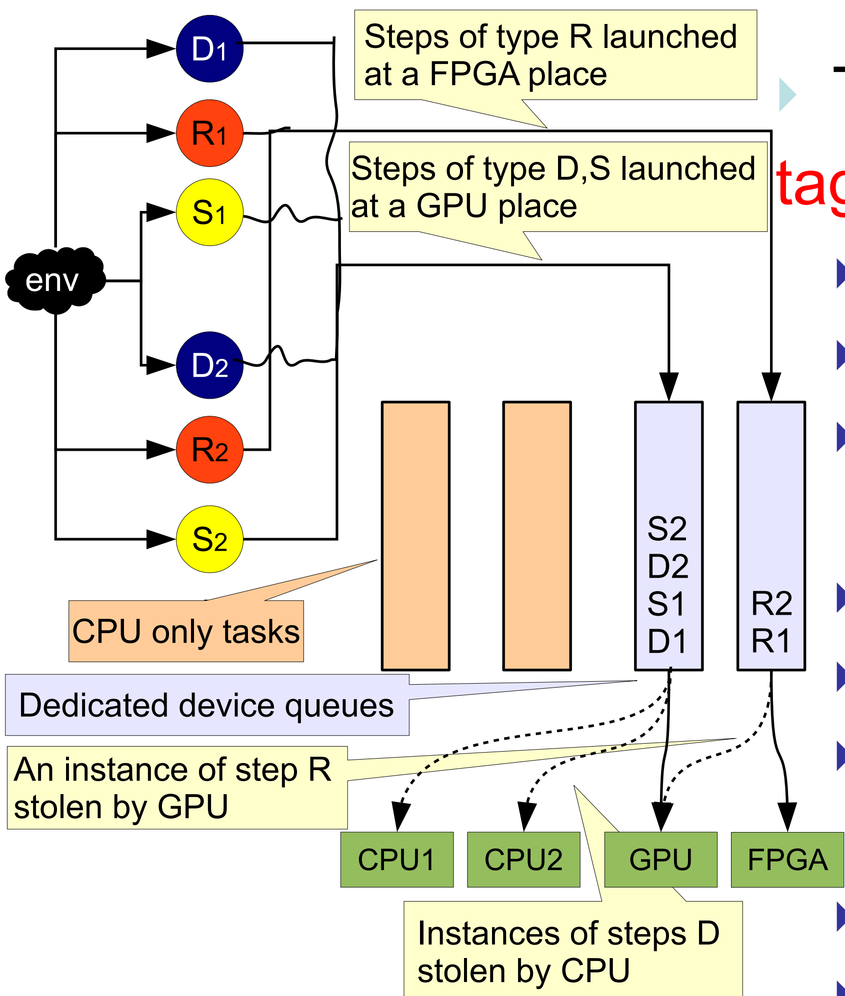
HPTs in Habanero-C



- Devices (GPU or FPGA) are represented as memory module places and agent workers
- Explicit data transfer between main memory and device memory
- Device tasks are created by a CPU worker via async (gpl)

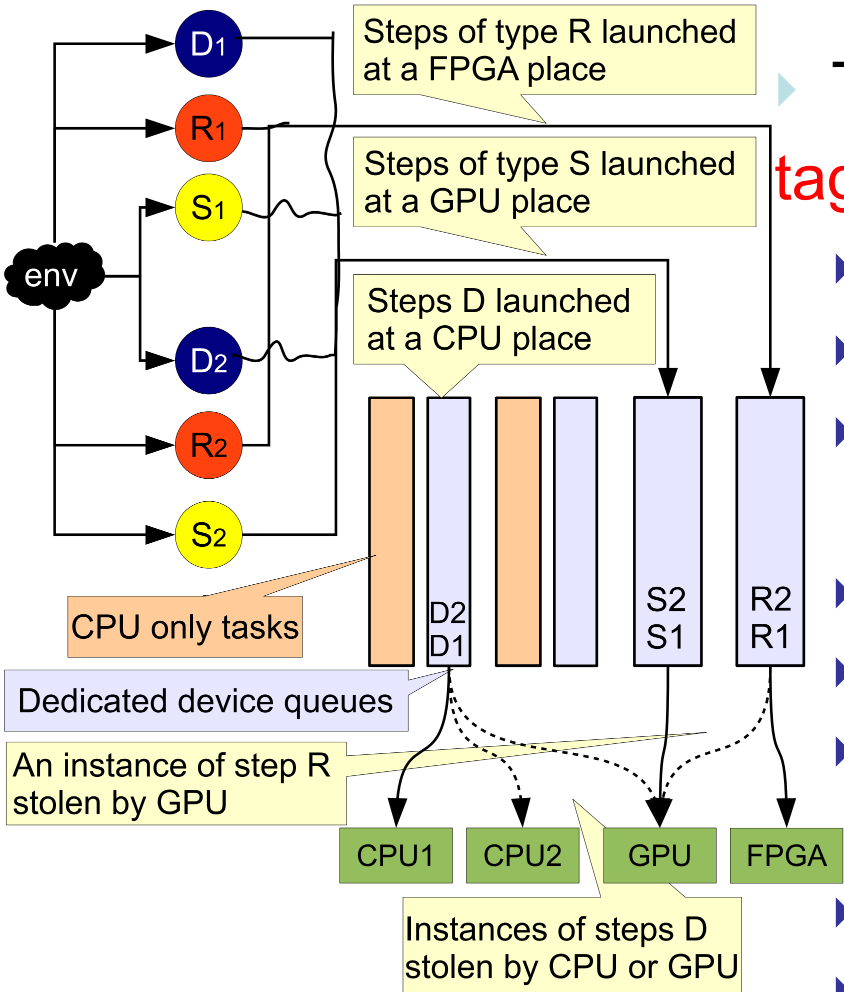
Slide credit: Habanero-C team

Habanero-C runtime



- Textual graph representation with tag functions and affinity annotations:
  - < C > :: (D @CPU=20,GPU=10);
  - < C > :: (R @GPU=5, FPGA=10);
  - < C > :: (S @GPU=12);
  - [IN : k-1] → (D : k) → [IN2 : k+1];
  - [IN2 : 2*k] → (R : k) → [IN3 : k/2];
  - [IN3 : k] → (S : k) → [OUT : IN3[k]];
  - env → [IN : { 0 .. 9 }], < C : { 0 .. 9 } >;
  - [OUT : 1] → env;

Habanero-C runtime



- Textual graph representation with tag functions and affinity annotations:
  - < C > :: (D @CPU=20,GPU=10);
  - < C > :: (R @GPU=5, FPGA=10);
  - < C > :: (S @GPU=12);

  - [IN : k-1] → (D : k) → [IN2 : k+1];
  - [IN2 : 2*k] → (R : k) → [IN3 : k/2];
  - [IN3 : k] → (S : k) → [OUT : IN3[k]];

  - env → [IN : { 0 .. 9 }], < C : { 0 .. 9 } >;
  - [OUT : 1] → env;

CnC-HC runtime

- Motivation: Data dependencies (*Gets*) needs extra synchronization beyond HC constructs:
  - *async* and *finish*

- Data Driven Runtime
  - Steps do not start to execute until all data is available
  - Dependencies are “filled in” when step is prescribed
  - Once all dependencies are satisfied, step executes =>

    Gets are ensured to succeed.

    (Read operations, auto-generated, transparent to user)

Language and runtime contributions

- Language extensions
  - Tag functions and Ranges
    - [input : {f1(k) .. f2(k)}] → (s1Step : k) → [output : g(k)]
    - Enable automatic generation of high-level operations
  - Step affinity
    - < tag1 > :: (s1Step : @CPU= 33,GPU= 9, FPGA= 5);
    - Auto-generate code to launch step at a device place
- Runtime contributions
  - Extend HC scheduler with cross-device work-stealing
  - Data Driven Runtime in CnC-HC

Outline

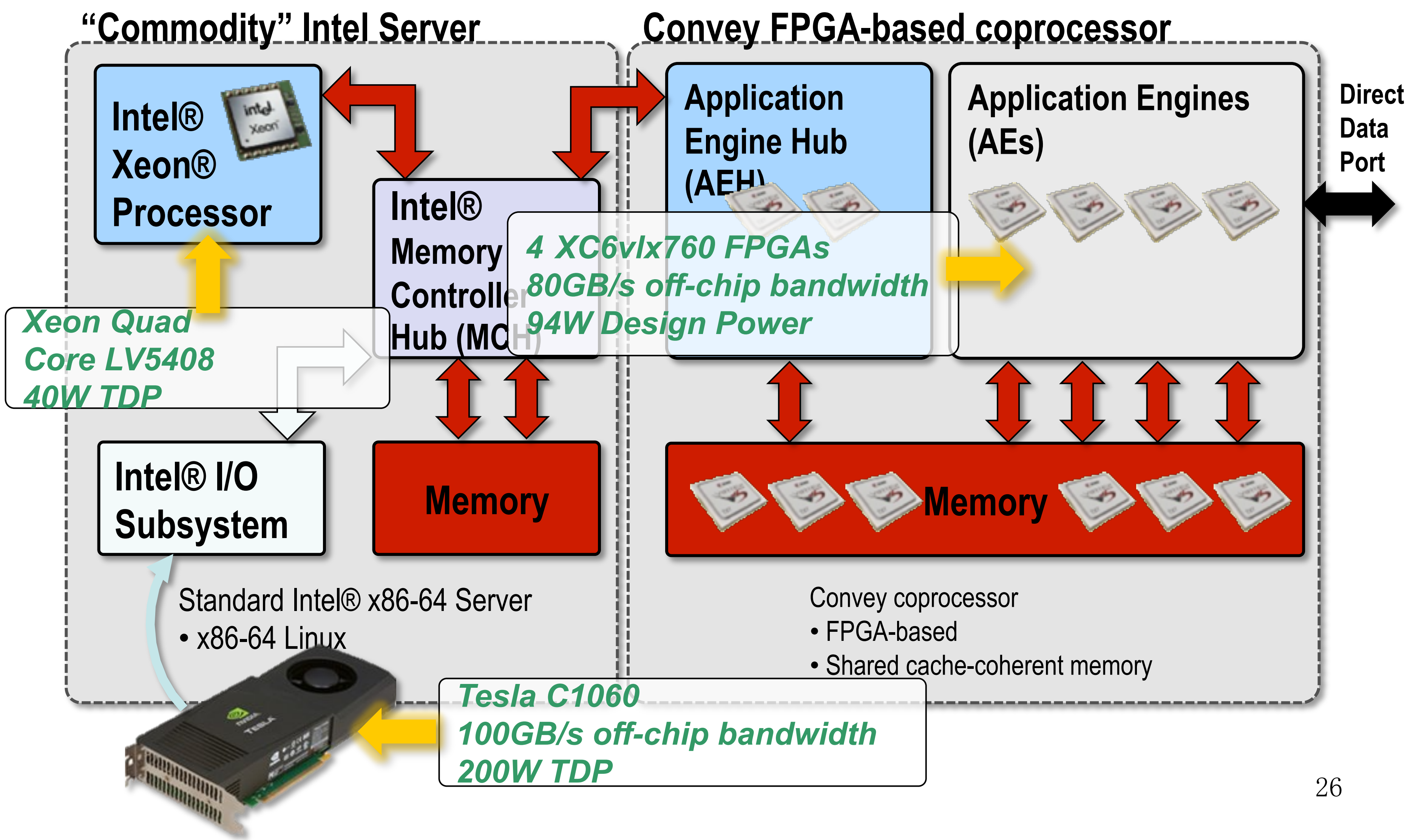
- Concurrent Collections programming model
- Programming model and runtime extensions
- Target platform
- Experimental results
- Conclusions and future work

Experimental setup

- Convey HC-1ex
  - 4 Xilinx Virtex6 LX760 - 80GB/s off-chip bandwidth
  - Xeon Quad Core LV5408
  - Tesla C1060 - 100GB/s off-chip bandwidth
  - 16GB capacity for coprocessor side memory
  - Shared memory model between CPU and FPGA (but not GPU)

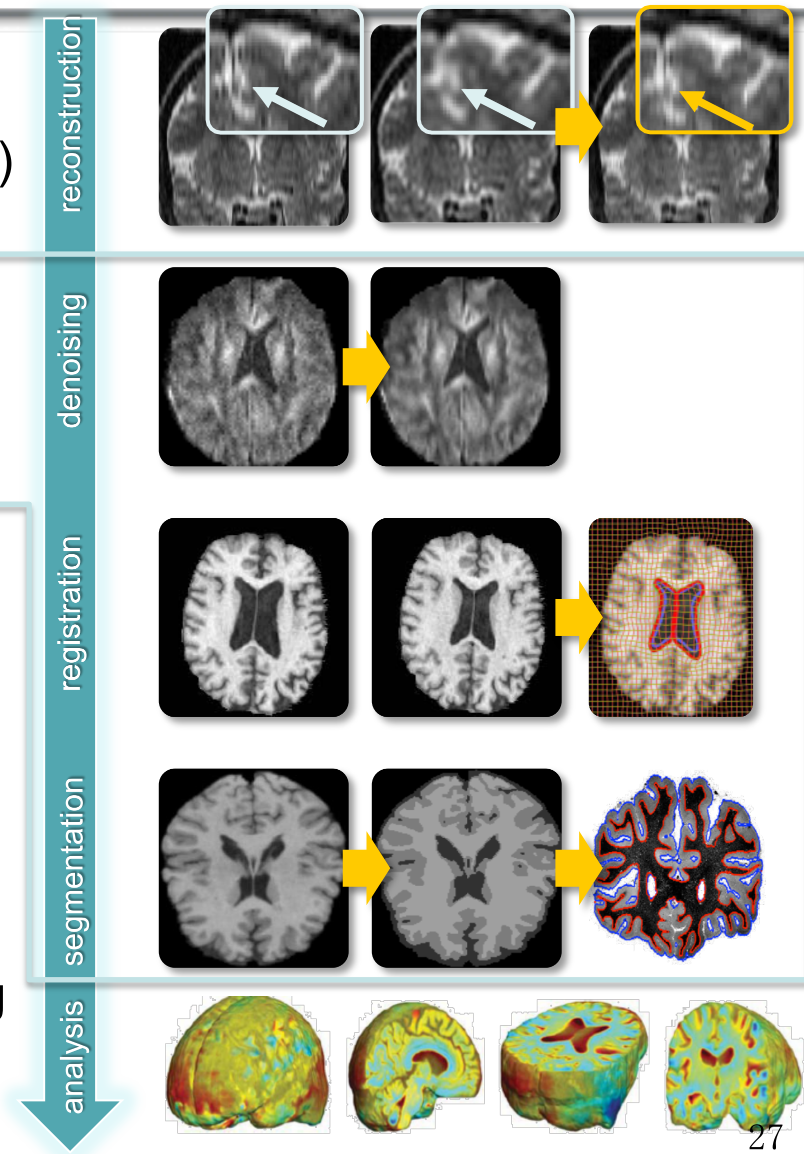
- Medical imaging pipeline – C and CUDA steps

HC-1ex architecture



Medical imaging application

- New reconstruction methods
  - decrease radiation exposure (CT)
  - number of samples (MR)
- 3D/4D image analysis pipeline
  - Denoising
  - Registration
  - Segmentation
- Analysis
  - Real-time quantitative cancer assessment applications
- Potential:
  - order-of-magnitude performance improvement
  - power efficiency improvements
  - real-time clinical applications and simulations using patient imaging data



Slide credit: Habanero team and NSF Expeditions CDSC project

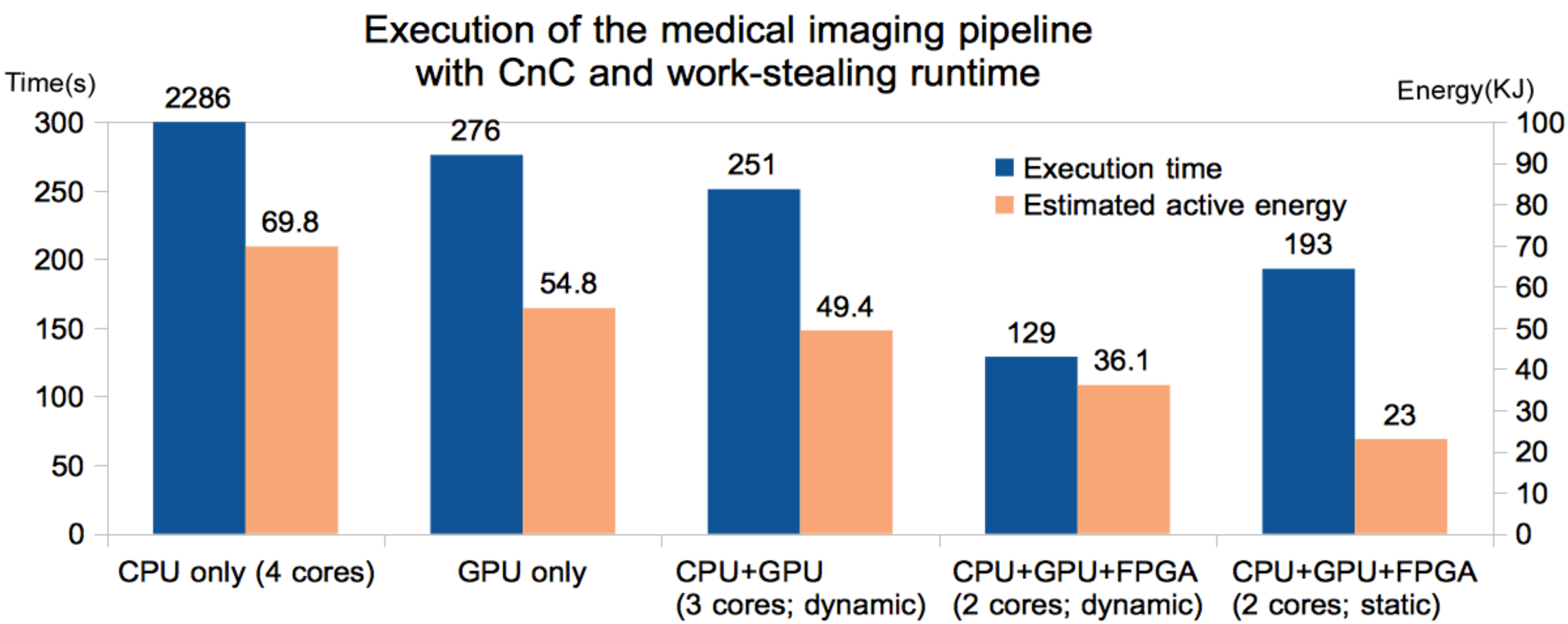
Experimental results

- Performance for medical imaging kernels

	Denoise	Registration	Segmentation
Num iterations	3	100	50
CPU (1 core)	3.3s	457.8s	36.76s
GPU	0.085s (38.3 ×)	20.26s (22.6 ×)	1.263s (29.1 ×)
FPGA	0.190s (17.2 ×)	17.52s (26.1 ×)	4.173s (8.8 ×)

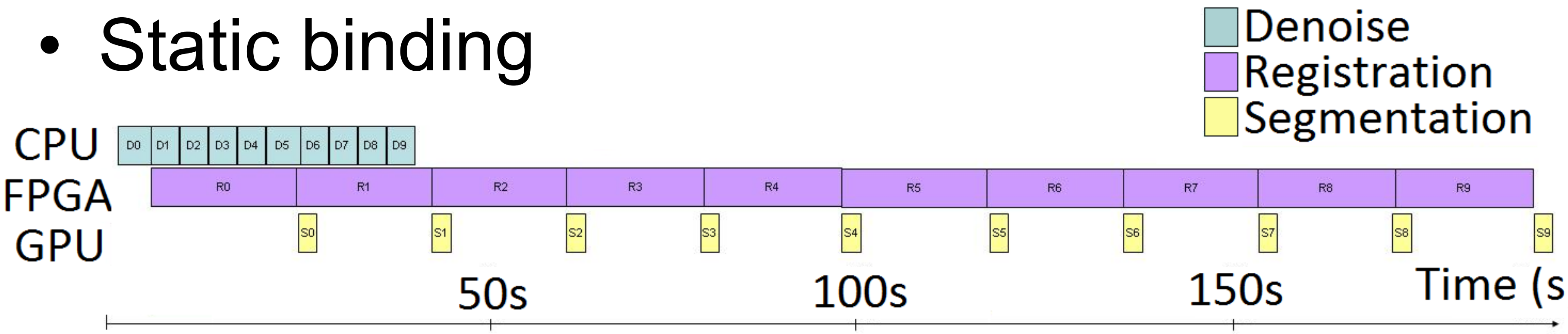
Experimental results

- Execution times and active energy with dynamic work stealing

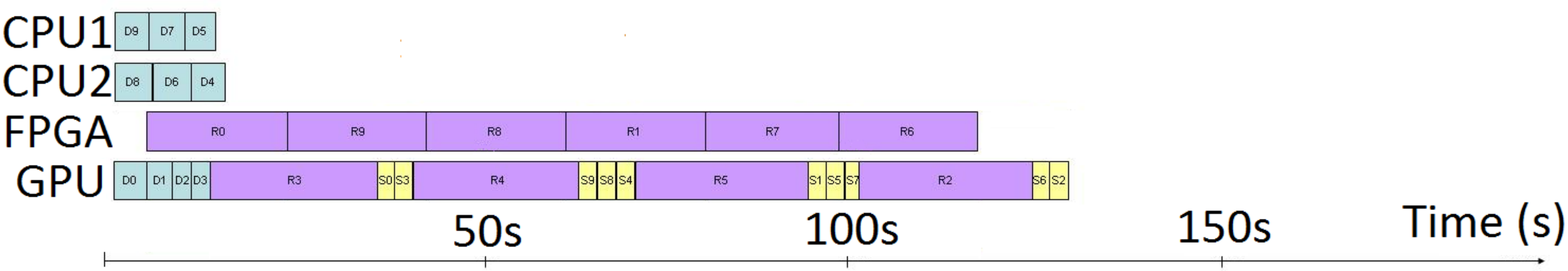


Static vs Dynamic binding

- Static binding



- Dynamic Binding



Conclusions

- Obtaining a hybrid execution model
  - Language extensions within the CnC model
  - Cross-device work-stealing using Habanero-C
  - High performance (17.72X speedup)
  - Low energy consumption (0.52X of the power used by a CPU)
  - Real-world medical image-processing pipeline
  - Unique prototype heterogeneous platform (CPU, GPU, FPGA)

Ongoing and future work

- Use tag functions for further graph analysis (correctness, memory optimizations)
- Determining the affinity metric at runtime (application fine tuning)
- Evaluate on more benchmarks
- Determine useful primitives for domain experts
  - Easy-to-program modeling software
  - Customizable hardware platform

Acknowledgements

- NSF Expeditions Center for Domain-Specific Computing (CDSC) --- UCLA, Rice, OSU, UCSB

  http://cdsc.ucla.edu

- Habanero-C (HC) team:

  https://wiki.rice.edu/confluence/display/HABANERO/Habanero-C+Programming+Language

- Habanero Multicore Software Research Project

  http://habanero.rice.edu

Thank you!

- Questions on CnC-HC ?
  - Language extensions within the CnC model
  - Cross-device work-stealing using Habanero-C
  - High performance (17.72X speedup)
  - Low energy consumption (0.52X of the power used by a CPU)
  - Real-world medical image-processing pipeline
  - Unique prototype heterogeneous platform (CPU, GPU, FPGA)